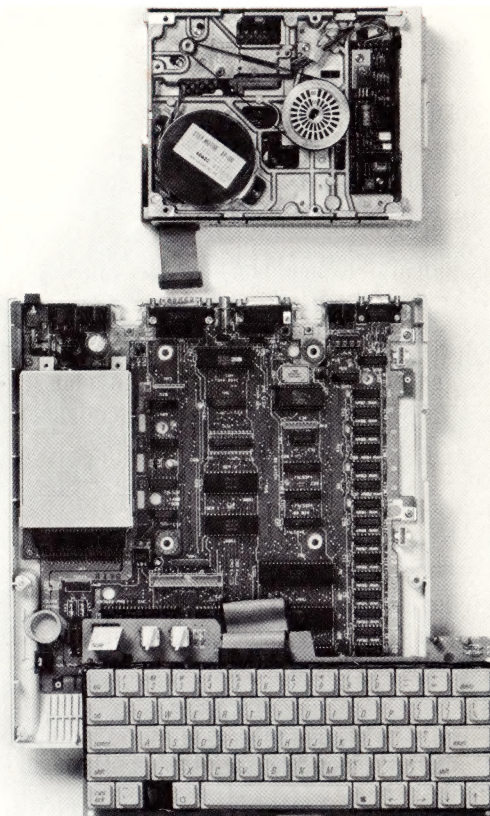




Apple® IIc Technical Reference Manual



Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts Menlo Park, California Don Mills, Ontario
Wokingham, England Amsterdam Sydney Singapore Tokyo
Madrid Bogotá Santiago San Juan

Copyright © 1985 by Apple Computer, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-17727-7

BCDEFGHIJ-DO-89876

Second Printing, February 1986



Apple® IIc Technical Reference Manual

Contents

Figures and Tables	xix
Radio and Television Interference Statement	xxix

PREFACE

About This Manual	xxxi
Contents of This Manual	xxxii
Symbols Used in This Manual	xxxiii

CHAPTER 1

Introduction	1
1.1 Outside of Machine	2
1.1.1 The Keyboard	3
Features	3
Special Function Keys	4
Cursor Movement Keys	4
Modifier Keys	5
The 80/40 Switch	5
The Keyboard Switch	5
Disk-Use and Power Lights	7
1.1.2 The Speaker	7
1.1.3 The Built-in Disk Drive	8
1.1.4 The Back Panel	8
1.2 Inside of Machine	10
1.2.1 The Internal Voltage Converter	10
1.2.2 The Main Logic Board	11
1.2.3 The Other Circuit Boards	13

2.1 The 65C02 Microprocessor	16
2.2 Overview of the Address Space	18
2.3 Memory Map and Memory Switching	18
2.3.1 Main RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)	20
2.3.2 Auxiliary RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)	20
2.3.3 ROM Addresses (\$C100-\$FFFF)	20
2.3.4 Hardware Addresses (\$C000-\$C0FF)	21
2.4 Bank-Switched Memory	22
2.4.1 Page Allocations	24
Page \$00 (One-Byte Addresses)	24
Page \$01 (The 65C02 Stack)	24
Pages \$D0-\$FF (ROM and RAM)	24
2.4.2 Using Bank Selector Switches	25
2.5 48K Memory	34
2.5.1 Page Allocations	34
Page \$02 (The Input Buffer)	34
Page \$03 (Global Storage and Vectors)	34
Pages \$04-\$07 (Text and Low-Resolution Page 1)	34
Pages \$08-\$0B (Text and Low-Resolution Page 2)	36
Page \$08 (Communication Port Buffers)	36
Pages \$20-\$3F (High-Resolution Page 1)	36
Pages \$40-\$5F (High-Resolution Page 2)	37
2.5.2 Using 48K Memory Switches	37
2.5.3 Transfers Between Main and Auxiliary Memory	40
Transferring Data	40
Transferring Control	41
2.5.4 Using Display Memory Switches	43

2.6 The Reset Routine	47
2.6.1 The Cold-Start Procedure (Power On)	49
2.6.2 The Warm-Start Procedure (CONTROL-RESET)	49
2.6.3 Forced Cold Start (OPEN APPLE-CONTROL-RESET)	49
2.6.4 The Reset Vector	50

CHAPTER 3

Introduction to Apple IIc I/O	51
3.1 The Standard I/O Links	52
3.2 Standard Input Features	54
3.2.1 RdKey Subroutine	54
3.2.2 KeyIn Subroutine	54
3.2.3 GetLn Subroutine	55
3.2.4 Escape Codes With GetLn	56
3.2.5 Editing With GetLn	58
Cancel Line	58
Backspace	58
Retype	59
3.3 Standard Output Features	59
3.3.1 COut Subroutine	59
3.3.2 Control Characters With COut1	60
3.3.3 Control Characters With C3COut1	61
3.3.4 The Stop-List Feature	62
3.3.5 The Text Window	63
3.3.6 Normal, Inverse, and Flashing Text	64
Primary Character Set Display	65
Alternate Character Set Display	65

3.4 Port I/O	66
3.4.1 Standard Link Entry Points	66
3.4.2 Firmware Protocol	67
3.4.3 Port I/O Space	68
3.4.4 Port ROM Space	69
3.4.5 Expansion ROM Space	69
3.4.6 Port Screen Hole RAM Space	69
3.5 Interrupts	70

CHAPTER 4

Keyboard and Speaker	71
4.1 Keyboard Input	72
4.1.1 Reading the Keyboard	72
4.1.2 Monitor Firmware Support for Keyboard Input	76
4.2 Speaker Output	77
4.2.1 Using the Speaker	77
4.2.2 Monitor Firmware Support for Speaker Output	78

CHAPTER 5

Video Display Output	79
5.1 Video Display Specifications	81
5.2 Text Modes	82
5.2.1 Text Character Sets	82
5.2.2 MouseText	83
5.2.3 40-Column Versus 80-Column Text	84
5.3 Graphics Modes	87
5.3.1 Low-Resolution Graphics	87
5.3.2 High-Resolution Graphics	88
5.3.3 Double-High-Resolution Graphics	91

5.4 Mixed-Mode Displays	91
5.5 Display Pages	93
5.6 Display Mode Switching	94
5.7 Display Page Maps	97
5.8 Monitor Support for Video Display Output	104
5.9 I/O Firmware Support for Video Display Output	108

CHAPTER 6

Disk Input and Output	111
6.1 Startup	113
6.2 External Drive Startup	114
6.3 The Protocol Converter	114
6.3.1 Locating the Protocol Converter	115
6.3.2 Issuing a Call to the Protocol Converter	115
6.3.3 Cautions	116
6.4 Descriptions of the Protocol Converter Calls	117
6.4.1 STATUS	119
6.4.2 READ BLOCK	123
6.4.3 WRITE BLOCK	124
6.4.4 FORMAT	126
6.4.5 CONTROL	127
6.4.6 INIT	130
6.4.7 OPEN	131
6.4.8 CLOSE	132
6.4.9 READ	133
6.4.10 WRITE	134
6.5 An Example: Issuing a Protocol Converter Call	136
6.6 Summary of Commands and Parameters	140
6.7 Summary of Error Codes	141

CHAPTER 7

Serial I/O Port 1

143

- 7.1 Using Serial Port 1 145
- 7.2 Characteristics of Port 1 at Startup 148
- 7.3 Hardware Page Locations for Port 1 149
- 7.4 I/O Firmware Support for Port 1 149
- 7.5 Screen Hole Locations for Port 1 150
- 7.6 Changing Port 1 Characteristics 151
 - 7.6.1 Data Format and Baud Rate 152
 - 7.6.2 Carriage Return and Line Feed 153
 - 7.6.3 Sending Special Characters 154
 - 7.6.4 Displaying Output on the Screen 154

CHAPTER 8

Serial I/O Port 2

155

- 8.1 Using Serial Port 2 157
- 8.2 Characteristics of Port 2 at Startup 160
- 8.3 Hardware Page Locations for Port 2 161
- 8.4 I/O Firmware Support for Port 2 161
- 8.5 Screen Hole Locations for Port 2 162
- 8.6 Changing Port 2 Characteristics 163
 - 8.6.1 Data Format and Baud Rate 164
 - 8.6.2 Carriage Return and Line Feed 166
 - 8.6.3 Routing Input and Output 166
 - Half-Duplex Operation 167
 - Full-Duplex Operation 168
 - Terminal Mode 171

CHAPTER 9

Mouse and Game Input

173

9.1 Mouse Input 174

9.1.1 Mouse Connector Signals 175

9.1.2 Mouse Operating Modes 175

Transparent Mode 175

Movement Interrupt Mode 175

Button Interrupt Mode 176

Movement/Button Interrupt Mode 176

Vertical Blanking Active Modes 176

9.1.3 Mouse Soft Switches 176

9.1.4 I/O Firmware Support for Mouse Input 179

Pascal Support 181

BASIC and Assembly-Language Support 182

9.1.5 Screen Holes 182

9.1.6 Using the Mouse as a Hand Controller 183

9.2 Game Input 184

9.2.1 The Hand Controller Connector Signals 185

Switch Inputs (Sw0 and Sw1) 185

Analog Inputs (Pdl0 and Pdl1) 186

9.2.2 Monitor Support for Game Input 186

CHAPTER 10

Using the Monitor

187

10.1 Invoking the Monitor 188

10.2 Syntax of Monitor Commands 189

10.3 Monitor Memory Commands	189
10.3.1 Examining Memory Contents	190
10.3.2 Memory Dump	190
10.3.3 Changing Memory Contents	192
Changing One Byte	192
Changing Consecutive Locations	193
10.3.4 Moving Data in Memory	194
10.3.5 Comparing Data in Memory	196
10.4 Monitor Register Commands	197
10.4.1 Changing Registers	197
10.4.2 Examining Registers	197
10.5 Miscellaneous Monitor Commands	198
10.5.1 Display Inverse and Normal	198
10.5.2 Back to BASIC	198
10.5.3 Redirecting Input and Output	199
10.5.4 Hexadecimal Arithmetic	199
10.6 Advanced Operations	200
10.6.1 Multiple-Command Lines	200
10.6.2 Filling Memory	200
10.6.3 Repeating Commands	201
10.6.4 Creating Your Own Commands	202
10.7 Machine-Language Programs	203
10.7.1 Running a Program	203
10.7.2 Disassembled Programs	204
10.8 The STEP and TRACE Commands	205
10.9 The Mini-Assembler	207
10.9.1 Starting the Mini-Assembler	208
10.9.2 Using the Mini-Assembler	208
10.9.3 Mini-Assembler Instruction Formats	210

10.10	Summary of Monitor Commands	212
10.10.1	Examining Memory	212
10.10.2	Changing the Contents of Memory	212
10.10.3	Moving and Comparing	213
10.10.4	The Register Command	213
10.10.5	Miscellaneous Monitor Commands	213
10.10.6	Running and Listing Programs	214

CHAPTER 11

Hardware Implementation		215
11.1	Environmental Specifications	216
11.2	Power Requirements	217
11.2.1	The External Power Supply	217
11.2.2	The External Power Connector	218
11.2.3	The Internal Converter	219
11.3	Apple IIc Overall Block Diagram	220
11.4	The 65C02 Microprocessor	220
11.4.1	65C02 Block Diagram	220
11.4.2	65C02 Timing	223
11.5	The Custom Integrated Circuits	225
11.5.1	The Memory Management Unit (MMU)	225
11.5.2	The Input/Output Unit (IOU)	227
11.5.3	The Timing Generator (TMG)	229
11.5.4	The General Logic Unit (GLU)	230
11.5.5	The Disk Controller Unit (IWM)	231

11.6	Memory Addressing	232
11.6.1	ROM Addressing	233
11.6.2	RAM Addressing	234
	Dynamic RAM Refreshment	235
	Dynamic RAM Timing	236
11.7	The Keyboard	237
11.8	The Speaker	240
11.8.1	Volume Control	240
11.8.2	Output Jack	240
11.9	The Video Display	241
11.9.1	The Video Counters	241
11.9.2	Display Memory Addressing	242
11.9.3	Display Address Mapping	242
11.9.4	Video Display Modes	246
	Text Displays	247
	Low-Resolution Display	250
	High-Resolution Display	251
	Double-High-Resolution Display	252
11.9.5	Video Output Signals	253
	Monitor Output	253
	Video Expansion Output	254
11.10	Disk I/O	256
11.11	Serial I/O	257
11.11.1	ACIA Control Register	261
11.11.2	ACIA Command Register	263
11.11.3	ACIA Status Register	264
11.11.4	ACIA Transmit/Receive Register	265
11.12	Mouse Input	265
11.13	Hand Controller Input	270
11.14	Schematic Diagrams	274

APPENDIX A	<hr/> The 65C02 Microprocessor	281
	A.1 Differences Between 6502 and 65C02	282
	A.1.1 Differing Cycle Times	282
	A.1.2 Differing Instruction Results	283
	A.2 Data Sheet	283
APPENDIX B	<hr/> Memory Map	293
	B.1 Page \$00	294
	B.2 Page \$03	298
	B.3 Screen Holes	298
	B.4 The Hardware Page	302
APPENDIX C	<hr/> Important Firmware Locations	309
	C.1 The Tables	310
	C.2 Port Addresses	311
	C.3 Other Video and I/O Firmware Addresses	313
	C.4 Applesoft BASIC Interpreter Addresses	313
	C.5 Monitor Addresses	313
APPENDIX D	<hr/> Operating Systems and Languages	315
	D.1 Operating Systems	316
	D.1.1 ProDOS	316
	D.1.2 DOS	316
	D.1.3 Pascal Operating System	316

D.2 Languages	317
D.2.1 Applesoft BASIC	317
D.2.2 Integer BASIC	317
D.2.3 Pascal	317
D.2.4 FORTRAN	317
D.2.5 Logo II	318

APPENDIX E

Interrupts	319
E.1 Introduction	320
E.1.1 What Is an Interrupt?	320
E.1.2 Interrupts on Apple II Computers	320
E.1.3 Interrupt Handling on the 65C02	321
E.1.4 The Interrupt Vector at \$FFFE	322
E.2 The Built-in Interrupt Handler	322
E.2.1 Saving the Memory Configuration	323
E.2.2 Managing Main and Auxiliary Stacks	324
E.3 User's Interrupt Handler at \$03FE	324
E.4 Handling Break Instructions	325
E.5 Sources of Interrupts	326
E.6 Firmware Handling of Interrupts	327
E.6.1 Firmware for Mouse and VBL	327
E.6.2 Firmware for Keyboard Interrupts	328
Using Keyboard Buffering Firmware	329
Using Keyboard Interrupts Through Firmware	329
E.6.3 Using External Interrupts Through Firmware	330

E.6.4 Firmware for Serial Interrupts	330
Using Serial Buffering Transparently	331
Using Serial Interrupts Through Firmware	331
Transmitting Serial Data	332
A Loophole in the Firmware	332
E.7 Bypassing the Interrupt Firmware	333
E.7.1 Using Mouse Interrupts Without the Firmware	333
E.7.2 Using ACIA Interrupts Without the Firmware	334

APPENDIX F

Apple II Series Differences	335
F.1 Overview	336
F.1.1 Type of Processor	337
F.1.2 Machine Identification	338
F.2 Memory Structure	338
F.2.1 Amount and Address Ranges of RAM	339
F.2.2 Amount and Address Ranges of ROM	339
F.2.3 Peripheral-Card Memory Spaces	340
F.2.4 Hardware Addresses	340
F.2.5 Monitors	343
F.3 I/O in General	344
F.3.1 DMA Transfers	344
F.3.2 Slots Versus Ports	344
F.3.3 Interrupts	344
F.4 Keyboard	345
F.4.1 Keys, Switches, and Lights	345
F.4.2 Character Sets	345
F.5 Speaker	346

F.6 Video Display	346
F.6.1 Character Sets	346
F.6.2 MouseText	347
F.6.3 Vertical Blanking	347
F.6.4 Display Modes	347
F.7 Disk I/O	348
F.8 Serial I/O	348
F.8.1 Serial Ports Versus Serial Cards	348
F.8.2 Serial I/O Buffers	349
F.9 Mouse and Hand Controllers	350
F.9.1 Mouse Input	350
F.9.2 Hand Controller Input and Output	350
F.10 Cassette I/O	351
F.11 Hardware	351
F.11.1 Power	352
F.11.2 Custom Chips	352

Appendix G

USA and International Models	353
G.1 Keyboard Layouts and Codes	354
G.1.1 USA Standard (Sholes) Keyboard	355
G.1.2 USA Simplified (Dvorak) Keyboard	358
G.1.3 ISO Layout of USA Keyboard	359
G.1.4 English Keyboard	360
G.1.5 French Keyboard	361
G.1.6 Canadian Keyboard	362
G.1.7 German Keyboard	364
G.1.8 Italian Keyboard	366
G.1.9 Western Spanish Keyboard	368

G.2 ASCII Character Sets 370

G.3 Certification 371

 G.3.1 Product Safety 371

 G.3.2 Important Safety Instructions 371

G.4 Power Supply Specifications 372

APPENDIX H

Conversion Tables 383

H.1 Bits and Bytes 374

H.2 Hexadecimal and Decimal 376

H.3 Hexadecimal and Negative Decimal 377

H.4 Peripheral Identification Numbers 378

H.5 Eight-Bit Code Conversions 380

APPENDIX I

Firmware Listings 385

Glossary 471

Bibliography 487

Index 489

Order Form

Tell Apple Card

Figures and Tables

CHAPTER 1

Introduction

1

Figure 1-1	Apple IIc External Features (Front)	2
Figure 1-2	Apple IIc External Features (Back)	2
Figure 1-3	Front of Apple IIc With Standard USA Keyboard	3
Table 1-1	Keyboard Specifications	4
Figure 1-4	USA Standard or Sholes Keyboard (Keyboard Switch Up)	6
Figure 1-5	USA Simplified or Dvorak Keyboard (Keyboard Switch Down)	6
Figure 1-6	Speaker, Volume Control, and Audio Output Jack	7
Figure 1-7	Built-in Disk Drive	8
Figure 1-8	Back Panel Connectors	9
Figure 1-9	Inside of Machine	10
Figure 1-10	Power Supply and Voltage Converter	11
Figure 1-11	Main Logic Board	12

CHAPTER 2

Memory Organization and Control

15

Figure 2-1	Internal Model of the 65C02 Microprocessor	17
Figure 2-2	Apple IIc Memory Map	19
Figure 2-3	Bank-Switched Memory Map	23
Table 2-1	Bank Selector Switches	26
Figure 2-4	Read ROM	27
Figure 2-5	Read ROM, Write RAM, and Use First \$D0 Bank	28
Figure 2-6	Read ROM, Write RAM, and Use Second \$D0 Bank	29
Figure 2-7	Read RAM and Use First \$D0 Bank	30
Figure 2-8	Read RAM and Use Second \$D0 Bank	31
Figure 2-9	Read and Write RAM and Use First \$D0 Bank	32

Figure 2-10	Read and Write RAM and Use Second \$D0 Bank	33
Figure 2-11	48K Memory Map	35
Table 2-2	48K Memory Switches	37
Figure 2-12	48K RAM Selection: Split Pairs	38
Figure 2-13	48K RAM Selection: One Side Only	39
Table 2-3	48K RAM Transfer Routines	40
Table 2-4	Parameters for MoveAux Routine	41
Table 2-5	Parameters for XFer Routine	42
Table 2-6	Display Memory Switches	43
Figure 2-14	Page2 Selections With 80Store On and HiRes Off	45
Figure 2-15	Page2 Selections With 80Store On and HiRes On	46
Figure 2-16	Reset Routine Flowchart	47
Table 2-7	Page \$03 Vectors	48

CHAPTER 3

Introduction to Apple IIc I/O

51

Table 3-1	Prompt Characters	55
Table 3-2	Escape Codes With GetLn	56
Table 3-3	Control Characters With COut1	60
Table 3-4	Control Characters With C3COut1	61
Table 3-5	Text Window Memory Locations	64
Table 3-6	Port Characteristics	66
Table 3-7	Firmware Protocol Locations	67
Table 3-8	Port I/O Locations	68
Table 3-9	Port Screen Hole Memory Locations	69

CHAPTER 4

Keyboard and Speaker

71

Table 4-1	Keyboard Input Characteristics	73
Table 4-2	Keys and ASCII Codes	74
Table 4-3	Speaker Output Characteristics	77

CHAPTER 5

Video Display Output

79

Table 5-1	Video Output Port Characteristics	80
Table 5-2	Video Display Specifications	81
Table 5-3	The Display Character Sets	83
Figure 5-1	MouseText Characters	84
Figure 5-2	40-Column and 80-Column Text (With Alternate Character Set)	85
Figure 5-3	Text Mode Characteristics and Switching	86
Table 5-4	Low-Resolution Graphics Colors	88
Figure 5-4	High-Resolution Display Bits	89
Table 5-5	High-Resolution Graphics Colors	90
Table 5-6	Double-High-Resolution Graphics Colors	92
Table 5-7	Video Display Page Locations	94
Table 5-8	Display Soft Switches	95
Table 5-9	Display Modes Supported by Firmware (Including Applesoft)	96
Table 5-10	Other Display Modes	97
Figure 5-5	Map of 40-Column Text Display	99
Figure 5-6	Map of 80-Column Text Display	100
Figure 5-7	Map of Low-Resolution Graphics Display	101

Figure 5-8	Map of High-Resolution Graphics Display	102
Figure 5-9	Map of Double-High-Resolution Graphics Display	103
Table 5-11	Monitor Firmware Routines	104
Table 5-12	Port 3 Firmware Protocol Table	108
Table 5-13	Pascal Video Control Functions	109

CHAPTER 6	Disk Input and Output	111
-----------	-----------------------	-----

Table 6-1	Disk I/O Port Characteristics	112
Figure 6-1	Summary of Protocol Converter Calls	140

CHAPTER 7	Serial I/O Port 1	143
-----------	-------------------	-----

Table 7-1	Serial Port 1 Characteristics	144
Table 7-2	Printer Port Commands	145
Table 7-3	Port 1 Hardware Page Locations	149
Table 7-4	Port 1 I/O Firmware Protocol	149
Table 7-5	Port 1 Screen Hole Locations	150
Figure 7-1	Diagram of Port 1 Characteristics Storage	151
Figure 7-2	Data Format	153

CHAPTER 8	Serial I/O Port 2	155
-----------	-------------------	-----

Table 8-1	Serial Port 2 Characteristics	156
Table 8-2	Modem Port Commands	157
Table 8-3	Port 2 Hardware Page Locations	161
Table 8-4	Port 2 I/O Firmware Protocol	162
Table 8-5	Port 2 Screen Hole Locations	162
Figure 8-1	Diagram of Port 2 Characteristics Storage	164

Figure 8-2	Devices in a Typical Communication Setup	165
Figure 8-3	Effect of IN#2	167
Figure 8-4	Effect of IN#2 and T Command (Half Duplex)	168
Figure 8-5	Effect of IN#2 and T Command (Full-Duplex Terminal)	169
Figure 8-6	Effect of IN#2, PR#2, and T Command (Full-Duplex Host)	170

CHAPTER 9

Mouse and Game Input 173

Table 9-1	Mouse Input Port Characteristics	174
Table 9-2	Mouse Soft Switches	177
Table 9-3	Mouse Firmware Routines	180
Table 9-4	Mouse Port I/O Firmware Protocol	181
Table 9-5	Mouse Port Screen Hole Locations	182
Table 9-6	Game Input Characteristics	184

CHAPTER 10

Using the Monitor 187

Table 10-1	Mini-Assembler Address Formats	211
------------	--------------------------------	-----

CHAPTER 11

Hardware Implementation 215

Table 11-1	Environmental Specifications	216
Table 11-2	Power Supply Specifications	218
Figure 11-1	External Power Connector	218
Table 11-3	External Power Connector Signals	218
Table 11-4	Internal Converter Specifications	219

Figure 11-2	Apple IIc Block Diagram	221
Figure 11-3	65C02 Block Diagram	222
Table 11-5	65C02 Microprocessor Specifications	223
Figure 11-4	65C02 Timing Signals	224
Table 11-6	65C02 Timing Signal Descriptions	224
Figure 11-5	The MMU Pinouts	226
Table 11-7	The MMU Signal Descriptions	226
Figure 11-6	The IOU Pinouts	227
Table 11-8	The IOU Signal Descriptions	227
Figure 11-7	The TMG Pinouts	229
Table 11-9	The TMG Signal Descriptions	229
Figure 11-8	The GLU Pinouts	230
Table 11-10	The GLU Signal Descriptions	230
Figure 11-9	The IWM Pinouts	231
Table 11-11	The IWM Signal Descriptions	231
Figure 11-10	Memory Bus Organization	233
Figure 11-11	The 23128 ROM Pinouts	234
Figure 11-12	The 2316 ROM Pinouts	234
Figure 11-13	The 2364 ROM Pinouts	234
Figure 11-14	The 64K RAM Pinouts	235
Table 11-12	RAM Address Multiplexing	236
Table 11-13	RAM Timing Signals	236
Figure 11-15	RAM Timing Signals	237
Figure 11-16	Keyboard Circuit Diagram	238
Figure 11-17	Keyboard Signals	239
Figure 11-18	Speaker Circuit Diagram	240
Figure 11-19	Display Address Transformation	243
Table 11-14	Display Memory Addressing	244

Table 11-15	Memory Address Bits for Display Modes	245
Figure 11-20	40-Column Text Display Memory	245
Figure 11-21	Video Display Circuits	246
Figure 11-22	7-MHz Video Timing Signals (40-Column, Low-Resolution, and High-Resolution Display)	248
Figure 11-23	14-MHz Video Timing Signals (80-Column and Double-High-Resolution Display)	249
Table 11-16	Character-Generator Control Signals	250
Figure 11-24	Video Output Back Panel Connectors	254
Figure 11-25	The Video Expansion Connector Pinouts	255
Table 11-17	The Video Expansion Connector Signals	255
Figure 11-26	Disk Drive Connector	257
Table 11-18	Disk Drive Connector Signals	257
Figure 11-27	Serial Port Circuits	258
Figure 11-28	6551 ACIA Block Diagram	259
Figure 11-29	The 6551 Pinouts	260
Table 11-19	The 6551 Signal Descriptions	260
Figure 11-30	Serial Port Connectors	261
Table 11-20	Serial Port Connector Signals	261
Figure 11-31	ACIA Control Register	262
Figure 11-32	ACIA Command Register	263
Figure 11-33	ACIA Status Register	264
Figure 11-34	Sample Mouse Waveform	265
Figure 11-35	Mouse Movement and Direction Waveforms	266
Figure 11-36	Mouse Connector	267
Table 11-21	Mouse Connector Signals	267
Figure 11-37	Mouse Circuits	268
Figure 11-38	Mouse Button Signals	269

Figure 11-39	Hand Controller Connector	270
Table 11-22	Hand Controller Connector Signals	271
Figure 11-40	How to Connect Switch Inputs	271
Figure 11-41	Hand Controller Circuits	272
Figure 11-42	Hand Controller Signals	273

APPENDIX A

The 65C02 Microprocessor281

Table A-1	Cycle Time Differences	282
-----------	------------------------	-----

APPENDIX B

Memory Map293

Table B-1	Page \$00 Use	294
Table B-2	Page \$03 Use	298
Table B-3	Main Memory Screen Hole Allocations	299
Table B-4	Auxiliary Memory Screen Hole Allocations	301
Table B-5	Addresses \$C000-\$C03F	303
Table B-6	Addresses \$C040-\$C05F	304
Table B-7	Addresses \$C060-\$C07F	305
Table B-8	Addresses \$C080-\$C0AF	306
Table B-9	Addresses \$C0B0-\$C0FF	307

APPENDIX C

Important Firmware Locations309

Table C-1	Serial Port 1 Addresses	311
Table C-2	Serial Port 2 Addresses	311
Table C-3	Video Firmware Addresses	312
Table C-4	Mouse Port Addresses	312

Table C-5	Apple IIc Enhanced Video and Miscellaneous Firmware	313
Table C-6	Apple IIc Monitor Entry Points and Vectors	313

APPENDIX E	Interrupts	319
	Figure E-1	Interrupt-Handling Sequence 323
	Table E-1	Activating Mouse Interrupts 333
	Table E-2	Reading Mouse Interrupts 334

APPENDIX F	Apple II Series Differences	335
	Table F-1	Apple II Series Identification Bytes 338
	Figure F-1	Apple II, II Plus, and IIe Hand Controller Signals 351

APPENDIX G	USA and International Models	353
	Figure G-1	USA Standard or Sholes Keyboard (Keyboard Switch Up) 355
	Table G-1	Keys and ASCII Codes 356
	Figure G-2	USA Simplified or Dvorak Keyboard (Keyboard Switch Down) 358
	Figure G-3	ISO Version of USA Standard Keyboard (Keyboard Switch Up) 359
	Figure G-4	English Keyboard (Keyboard Switch Down) 360
	Table G-2	English Keyboard Code Differences From Table G-1 360
	Figure G-5	French Keyboard (Keyboard Switch Down) 361

Table G-3	French Keyboard Code Differences From Table G-1	362
Figure G-6	Canadian Keyboard (Keyboard Switch Down)	363
Table G-4	Canadian Keyboard Code Differences From Table G-1	363
Figure G-7	German Keyboard (Keyboard Switch Down)	364
Table G-5	German Keyboard Code Differences From Table G-1	365
Figure G-8	Italian Keyboard (Keyboard Switch Down)	366
Table G-6	Italian Keyboard Code Differences From Table G-1	367
Figure G-9	Western Spanish Keyboard (Keyboard Switch Down)	368
Table G-7	Western Spanish Keyboard Code Differences From Table G-1	369
Table G-8	ASCII Code Equivalents	370
Table G-9	50-Hz Power Supply Specifications	372

APPENDIX H	Conversion Tables	375
Table H-1	What a Bit Can Represent	375
Figure H-1	Bits, Nibbles, and Bytes	375
Table H-2	Hexadecimal/Decimal Conversion	376
Table H-3	Hexadecimal to Negative Decimal Conversion	377
Table H-4	PIN Numbers	379
Table H-5	Control Characters, High Bit Off	381
Table H-6	Special Characters, High Bit Off	382
Table H-7	Uppercase Characters, High Bit Off	383
Table H-8	Lowercase Characters, High Bit Off	384

Radio and Television Interference

The equipment described in this manual generates and uses radio-frequency energy. If it is not installed and used properly—that is, in strict accordance with our instructions—it may cause interference with radio and television reception.

This equipment has been tested and complies with the limits for a Class B computing device in accordance with the specifications in Subpart J, Part 15, of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation, especially if a “rabbit ear” television antenna (the telescoping-rod type) is used.

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripherals. To further isolate the problem, disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it was caused by either the peripheral device or the I/O cable. These devices usually require shielded I/O cables. For Apple peripherals, you can obtain the proper shielded cable from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

If your computer does cause interference to radio or television reception, you can try to correct the interference by doing one or more of the following:

- ☐ Turn the television or radio antenna until the interference stops.
- ☐ Move the computer to one side or the other of the television or radio.
- ☐ Move the computer farther away from the television or radio.
- ☐ Plug the computer into an outlet that is on a different circuit than the television or radio. (That is, make certain the computer and the radio or TV are on circuits controlled by different circuit breakers or fuses.)
- ☐ Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and television.

If necessary, you should consult your Apple-authorized dealer or an experienced radio/television technician for additional suggestions.

This is the reference manual for the Apple® IIc personal computer. You will find a description here of all the hardware and firmware components of the Apple IIc.

The information in this manual is aimed at assembly-language programmers and hardware designers, but others interested in the internal operation of the Apple IIc can also benefit from reading it.

This manual tells you how the Apple IIc works, but not how to use the Apple IIc. If you need to know this, you should read other Apple IIc manuals, particularly the *Apple IIc Owner's Manual*.

This manual describes two versions of the Apple IIc: the original Apple IIc and a newer version that supports UniDisk™ 3.5. Information specific to the newer version of the machine is marked throughout the manual with **UniDisk 3.5** in the margin of the text. If you buy a UniDisk 3.5 for your original IIc, your dealer will install a new ROM to make your IIc equivalent to the newer version.

The following features have been added to the built-in firmware in the newer IIc:

- the Protocol Converter (Chapter 6)
- some new serial port commands (Chapters 7 and 8)
- the Mini-Assembler (Chapter 10)
- new Monitor commands STEP and TRACE (Chapter 10)

Changes have also been made to the built-in interrupt and mouse-interrupt handlers (Chapter 9 and Appendix E) and external drive startup procedures (Chapter 6).

Which Version Do You Have? You can quickly tell which version of the Apple IIc you have by checking the value of ROM location 64447 (\$FBFF hexadecimal). From Applesoft type **PRINT PEEK(64447)**. The value printed is 255 if your Apple IIc is an original version, and 0 if your Apple IIc supports UniDisk 3.5.

Contents of This Manual

The Apple IIc is presented here from the outside in.

Chapter 1 introduces the major physical features of the Apple IIc, including external controls and connectors and the major internal components of the machine.

Chapter 2 introduces the 65C02 microprocessor, which is the heart of the Apple IIc. It discusses the processor's address space, what resides in that space, and how to control it.

Chapter 3 introduces the I/O characteristics of the Apple IIc. Chapters 4 through 9 cover specific I/O devices.

- Chapter 4 describes the keyboard and speaker.
- Chapter 5 describes the video display.
- Chapter 6 describes the disk drive and the Protocol Converter.
- Chapter 7 describes serial port 1, generally used to control printers.
- Chapter 8 describes serial port 2, generally used to control modems.
- Chapter 9 describes the mouse, game paddle, and joystick port.

Chapter 10 covers the Monitor built into the Apple IIc's firmware. The Monitor helps you write, disassemble, and debug machine-language programs and inspect and manipulate Apple IIc memory contents.

Chapter 11 describes in detail the Apple IIc's hardware. It is intended to help programmers, but others may find the description useful.

Appendix A describes the differences between the 6502 used on most members of the Apple II family and the 65C02 used by the Apple IIc and enhanced Apple IIe. Most of the chapter is a reprint of the manufacturer's data sheet for the 65C02.

Appendix B contains a memory map of the Apple IIc including detailed maps of page \$00, page \$03, the screen holes, and the hardware page.

Appendix C lists the published entry points of the Apple IIc's firmware. The list, arranged by address, includes the I/O firmware (\$C300 through \$CFFF) and the Monitor firmware (\$F000 through \$FFFF). Addresses for the Applesoft interpreter firmware (\$D000 through \$F7FF) are listed in the *Applesoft BASIC Programmer's Reference Manual*.

Appendix D discusses some of the most common operating systems and languages that run on the Apple IIc, and what special features of the machine they do or don't use.

Appendix E describes how the Apple IIc's interrupt firmware operates and how to use it in your own programs.

Appendix F outlines the differences and similarities between the different members of the Apple II family.

Appendix G describes the keyboard layouts, code conversion tables, and external power requirements of USA and international models of the Apple IIc.

Appendix H contains tables to aid you in code and number base conversions.

Appendix I is a listing of the source code of the firmware of the version of the IIc that supports UniDisk 3.5. If you find that you absolutely must develop software or hardware for an original IIc that will not be upgraded in the future, you can get a listing of the old source code by filling out and mailing the order form in the back of this manual.

The glossary defines many of the technical terms used in this manual.

The bibliography lists articles and books with additional information about the Apple IIc and related products.

Finally, after the index at the back of this manual there is a *Tell Apple Card*. Please fill it out and send it in. Your experience with the manual can help us plan new reference materials.

Symbols Used in This Manual

This manual uses a numbering system to make it easier to cross-reference information. A reference like 2.4.6 means Chapter 2, Section 4, subsection 6. A reference like F.6.2 refers to Appendix F, Section 6, subsection 2.

Look for these visual cues throughout the manual:







- ▲Warning** | Important information about your safety or the safety of your data appears like this.
- Important!** | Text set off like this—and with a tag in the margin—presents important information.
- UniDisk 3.5** | Text set off like this presents information specific to the newer IIc, which supports UniDisk 3.5.

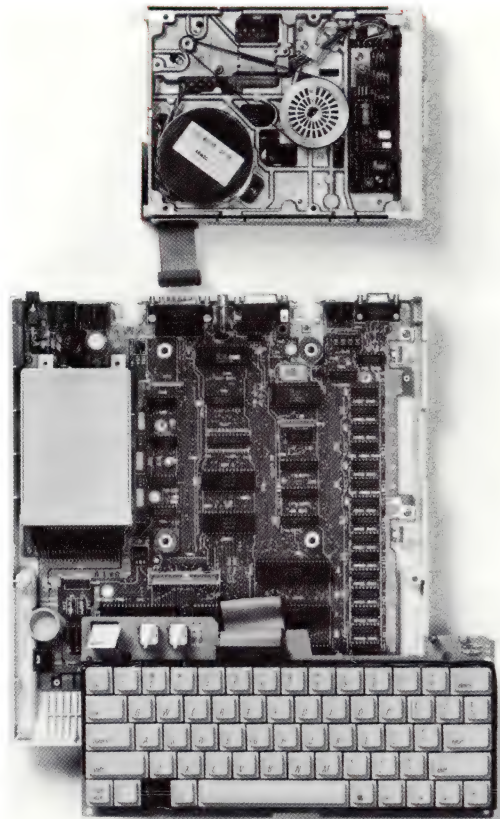
By the Way: Text set off this way presents incidental information that you may find interesting.

You will also see a special typeface used for what you type or for text that appears on your video display:

It looks like this.

Cross-references and definitions appear in marginal notes like this.

Keys look like this: , , . When you see a hyphen joining two keys, it means to press them simultaneously. For instance, -- means all three keys should be pressed at the same time.



This chapter introduces you to the working parts of the Apple® IIc by briefly describing the major components of the computer—both internal and external hardware and firmware—and telling you where in the manual to find out more about them.

1.1 Outside of Machine

This section briefly describes the Apple IIc's keyboard, controls, indicators, and expansion connectors.

The Apple IIc comes equipped with a keyboard, speaker (with audio output jack and volume control), built-in disk drive, external power supply, and internal voltage converter. It also has built-in interfaces with external connectors for a serial printer, video monitor, special video display adapters, modem, mouse, and game controllers. These external connectors allow you to plug in accessory equipment without having to go inside the machine to use expansion slots like those in the Apple IIe.

Figure 1-1 shows the front and right side of an Apple IIc, and Figure 1-2 shows the back and left side.

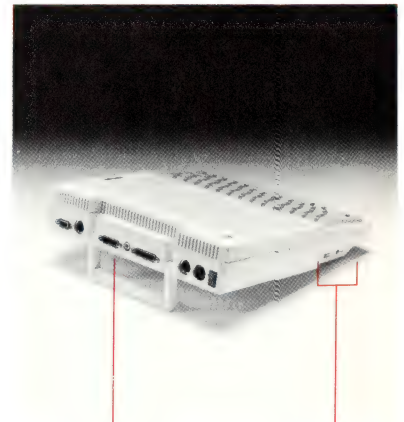
Figure 1-1. Apple IIc External Features (Front)



Keyboard
(See Figs. 1-4 and 1-5)

Disk Drive
(See Fig. 1-7)

Figure 1-2. Apple IIc External Features (Back)



Back Panel
(See Fig. 1-8)

Speaker
Volume Control
(See Fig. 1-6)

1.1.1 The Keyboard

ASCII stands for American Standard Code for Information Interchange. Table 4-2 lists the ASCII character encoding for the standard and simplified USA keyboards. Appendix G lists the encoding for international keyboards.

The Apple IIc's primary input device is the keyboard, shown in Figure 1-3. The keyboard has a 63-key typewriter layout with both uppercase and lowercase characters and can generate all 128 standard ASCII characters. A reset key, 80/40-column display selector switch, keyboard layout selector switch, disk-use light, and power light are also located on the front of the computer.

Figure 1-3. Front of Apple IIc With Standard USA Keyboard









Table 1-1, on the next page, lists the characteristics of all Apple IIc keyboards and front panels.

Features

The Apple IIc keyboard has automatic repeat on all character keys. This means that if you hold the key down longer than about a second, the character it generates repeats until you let up the key. It also has two-key rollover, which means if you press a key before releasing the one you pressed before it, the second character enters the computer the same as though you had released the previous key first. (This is important for fast touch-typists.)


Table 1-1. Keyboard Specifications

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Features:	Automatic repeat, two-key rollover
Special function keys:	RESET,  , 
Cursor movement keys:	 ,  ,  ,  , RETURN, DELETE, TAB
Modifier keys:	CONTROL, SHIFT, CAPS LOCK, ESC
Front panel switches:	80/40 switch, keyboard switch
Front panel lights:	Power light, disk-use light

Special Function Keys



The Apple IIc keyboard has three special function keys: RESET, and two keys marked with apples—one outlined, , and one filled in, .

RESET has a direct line to the 65C02 microprocessor’s RESET signal line (see Chapter 11): holding down CONTROL while pressing RESET causes the Apple IIc to restart processing with an internal firmware program that puts the machine in a known state (see Chapter 2).

You can restart the Apple IIc without turning the power off and back on again, by holding down both CONTROL and  while pressing RESET. Restarting this way is less stressful to the Apple IIc’s components than normal powerup.

Cursor Movement Keys

The Apple IIc keyboard has four cursor movement keys with arrows marked on them: left, right, down, and up. Three other keys can also cause cursor movements: RETURN, DELETE, and TAB. All seven of these keys generate ASCII control characters (see Table 4-2). It is up to the operating system or application program to interpret and act on the control codes that these keys generate.

The  and  keys are connected to 1-bit addresses in memory, described in Chapter 9.

Chapter 2 describes the results of the various reset procedures.

The **Monitor** is a built-in program that performs some of the basic activities of the computer, such as retrieving and storing key codes as they come in, and clearing or updating the display screen.

Modifier Keys

Three special keys—**CONTROL**, **SHIFT**, and **CAPS LOCK**—generate no codes when pressed by themselves, but change the codes generated by other keys they are pressed in combination with. A fourth key, **ESC**, generates a nonprinting control code that causes the Monitor to interpret certain subsequent keystrokes in a modified way.

- **CONTROL**, when pressed in combination with letter keys or certain other keys, produces ASCII control characters. Most of the control characters are invisible most of the time.
- **SHIFT** works the same on the Apple IIc as on an ordinary typewriter: it selects uppercase letters and the upper characters on the keys.
- **CAPS LOCK**, in its down position, changes the letter keys to uppercase, but does not affect other keys.
- **ESC** is not a modifier key in the same sense as **CONTROL** and **SHIFT**: you do not hold it down while pressing other keys. Rather, you press **ESC** and it generates the ASCII escape (ESC) control character (key code \$1B—see Table 4-2). When the **ESC** key is pressed, many programs—including the built-in Monitor program—then interpret other specific keys as designating an escape sequence.

The 80/40 Switch

The 80/40 switch lets you specify whether a program should display information in 40 or 80 columns per line. The switch indicates 40-column display when in its down position, and 80-column display when in its up position.

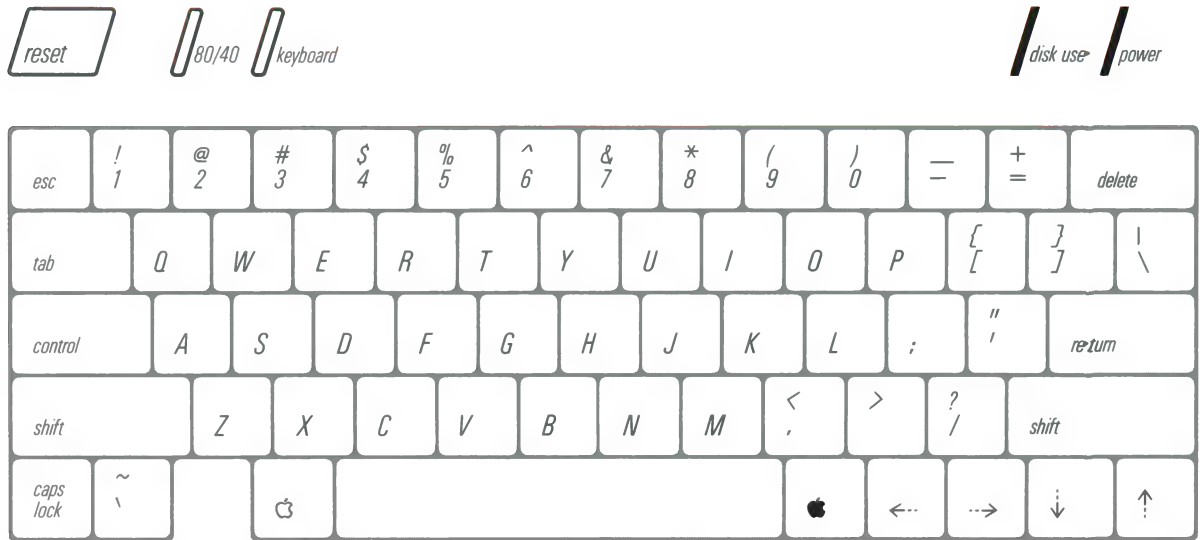
Important!

Not all programs check this switch. Even programs that do check the switch may do so only when the program first starts up. If that is the case, changing the switch position while the program is running will have no effect on the program's display. (See Table 4-1.)

The Keyboard Switch

You use the keyboard switch to select for use one of the two keyboard layouts and screen character sets built into your Apple IIc. On USA versions of the Apple IIc, you select the standard Sholes keyboard layout (Figure 1-4) with the switch in the up position, and the Dvorak simplified layout (Figure 1-5) with the switch in the down position.

If you normally use the Dvorak keyboard layout, you can *gently* pry up the keys from the keyboard and rearrange and replace them in their Dvorak positions.



Note: Shaded characters may be in different positions on some models.



Appendix G illustrates the keyboard layouts for both keyboard switch positions on several international versions of the Apple IIc.

On international models, the keycaps indicate the character positions for the local keyboard layout, which is selected when the keyboard switch is down. When up, the keyboard switch selects the USA standard characters and key layout.

Disk-Use and Power Lights

The red disk-use light glows whenever the built-in disk drive’s motor is switched on.

The green power light glows when the Apple IIc is turned on and normal power is present at the Apple IIc’s internal power supply.

▲Warning

If the power light flashes on and off, turn off the computer **immediately**. Find out what caused the condition (such as a brownout or short circuit) and fix the problem before turning the computer on again. Above all, do not use the disk drive when the power light is flashing; this may damage the computer.

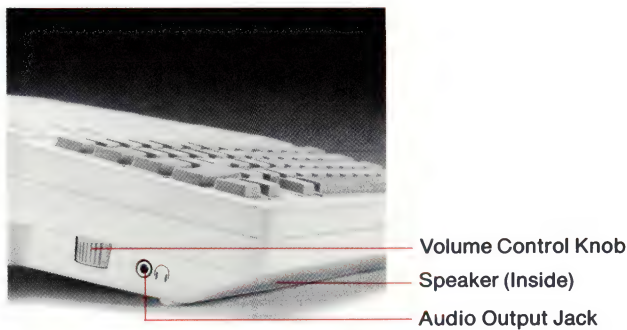
1.1.2 The Speaker

The red disk-use light glows whenever the built-in disk drive’s motor is switched on. The green power light glows when the Apple IIc is turned on and normal power is present at the Apple IIc’s internal power supply.

The way programs control the speaker is described in Section 4.2.

The Apple IIc has a speaker in the bottom of the case, as shown in Figure 1-6. The speaker lets Apple IIc programs produce a variety of sounds. There is also a volume control on the left side of the Apple IIc case, and a jack for connecting headphones or an external speaker. The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker.

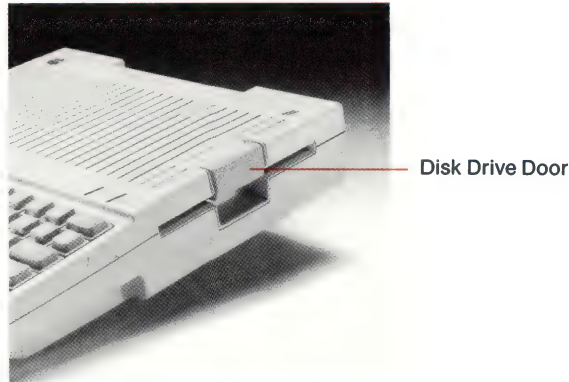
Figure 1-6. Speaker, Volume Control, and Audio Output Jack



1.1.3 The Built-in Disk Drive

The Apple IIc's built-in disk drive (Figure 1-7) is fully compatible with the Apple Disk IIc® that reads and writes 5¼-inch single-sided 35-track disks. The drive door is on the right side of the Apple IIc case.

Figure 1-7. Built-in Disk Drive



1.1.4 The Back Panel

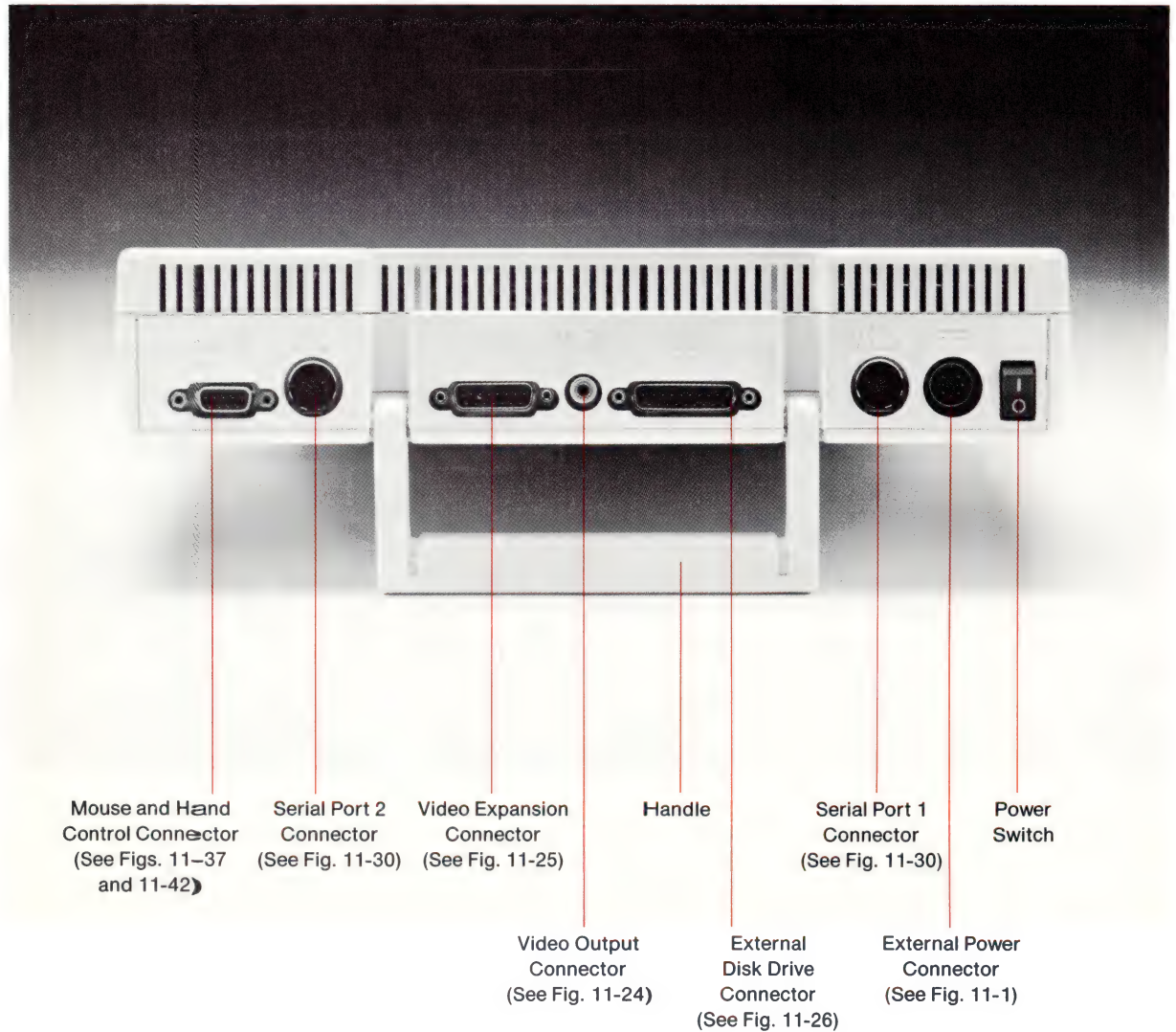
The back panel of the Apple IIc (Figure 1-8) has seven connectors and a main power switch. From left to right they are

- a 9-pin D-type miniature connector for connecting hand controllers, a mouse, a joystick, or some other device (see Chapters 9 and 11)
- a 5-pin DIN connector for serial input and output (port 2; normally for a modem) (see Chapters 7 and 11)
- a 15-pin D-type connector for video expansion (see Chapter 11)
- an RCA-type jack for a video monitor (see Chapter 11)
- a 19-pin D-type connector for connecting one or more external devices, such as intelligent disk drives (see Chapters 6 and 11)
- another 5-pin DIN connector for serial input and output (port 1; normally for a printer or plotter) (see Chapters 8 and 11)
- a special 7-pin DIN connector for power input (see Chapter 11).

Before attaching cables to the Apple IIc back panel connectors, be sure to move the handle until it clicks into position for propping up the computer. The handle should be down whenever the computer is running so that it can maintain proper cooling airflow.

The installation manuals for external devices contain instructions for connecting them to the Apple IIc.

Figure 1-8. Back Panel Connectors

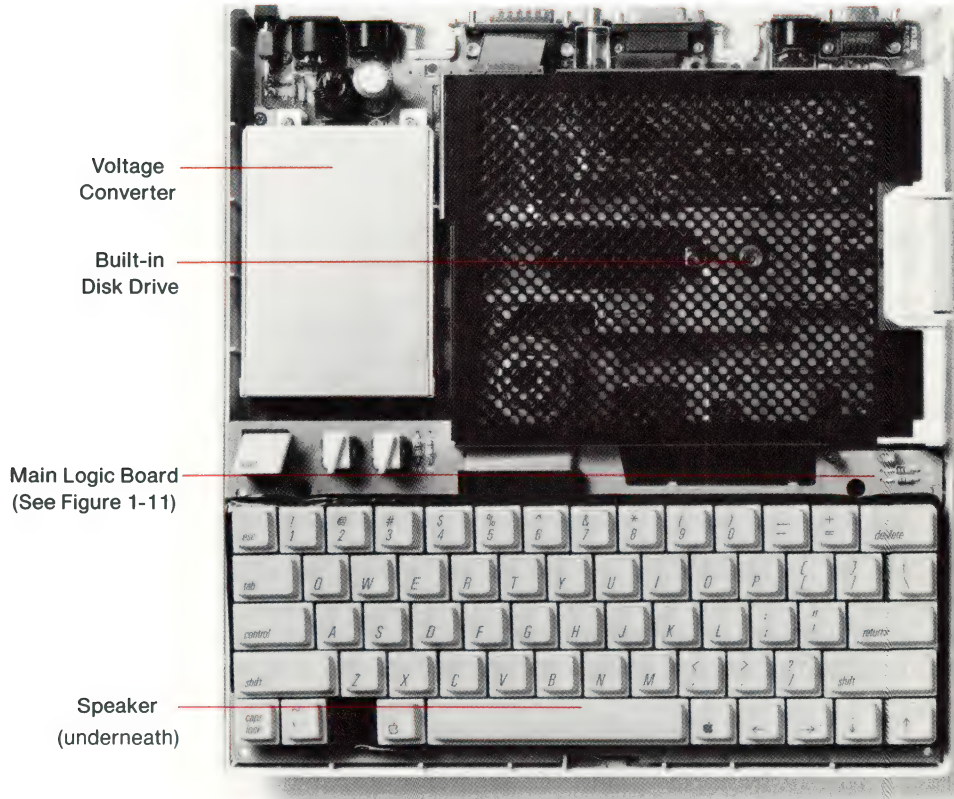


1.2 Inside of Machine

Chapter 11 discusses in further detail these components and how they work.

Figure 1-9 shows the main components inside the Apple IIc computer.

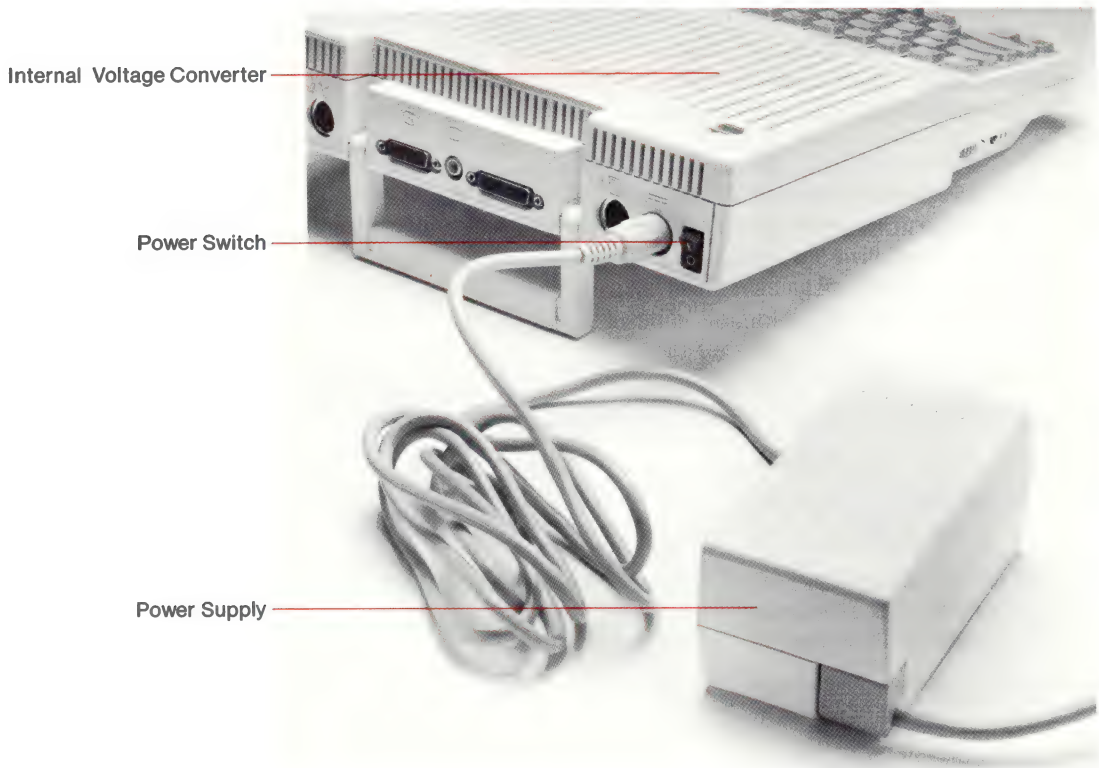
Figure 1-9. Inside of Machine



1.2.1 The Internal Voltage Converter

The built-in voltage converter operates from a 12 to 15 VDC input source, such as provided by the external power supply furnished with the Apple IIc (Figure 1-10). The voltage converter provides power for the logic board, built-in disk drive, one external disk drive, and the I/O signals available at the back panel.

Figure 1-10. Power Supply and Voltage Converter



Complete specifications of the Apple IIc power supply and voltage converter appear in Chapter 11.

The voltage converter produces three different voltages: +5V, +12V, and -12V. (Minus 5V, needed by some components in the Apple IIc, is derived from -12V on the main logic board.) It is a high-efficiency switching converter that protects itself and the rest of the Apple IIc against short circuits and other electrical mishaps.

1.2.2 The Main Logic Board

The main logic board, which is mounted flat in the bottom of the Apple IIc's case, has almost all the electronic parts of the computer attached to it.

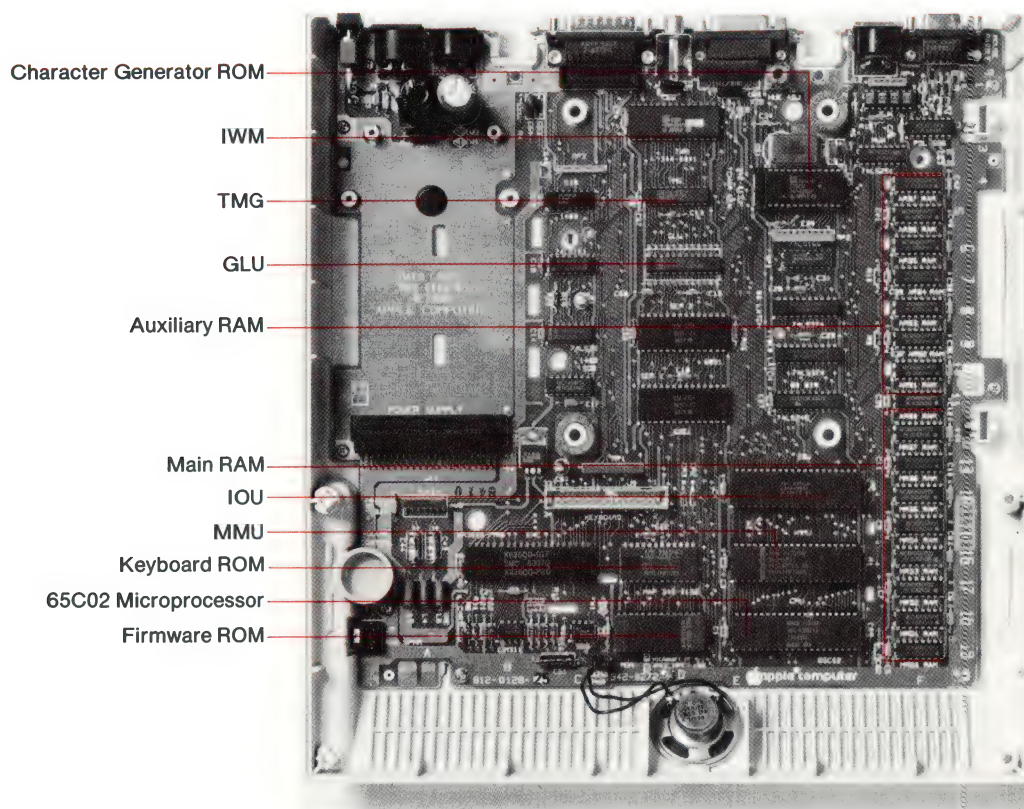
Firmware is program code that is stored in ROM. It can be read and executed, but not changed.

The specifications of the 65C02 are given in Chapter 11; the 65C02 instruction set is given in Appendix A.

Figure 1-11 shows the main logic board and the most important integrated circuits (ICs) in the Apple IIc. They are the CPU (central processing unit), RAM (random-access memory), ROM (read-only memory) ICs for keyboard encoding, display character generation, and firmware, and the five custom integrated circuits.

The processor is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502 used in other members of the Apple II family. It is an 8-bit microprocessor with a 16-bit address bus. In the Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 8-bit operations per second.

Figure 1-11. Main Logic Board



The Applesoft language interpreter is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.

Memory addressing: See Chapter 2.

See Chapters 3 through 9.

Chapter 11 discusses the functions of these integrated circuits in some detail.

The keyboard is scanned by an IC that generates matrix values for a ROM. The value of the ASCII code supplied by the ROM is latched at a specified memory location and is readable by programs.

The character generator ROM converts ASCII character values to a form that the video display can use.

The other ROM contains the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. The firmware that this ROM contains is described throughout this manual.

Five of the large IC's on the main logic board are custom-made for the Apple IIc:

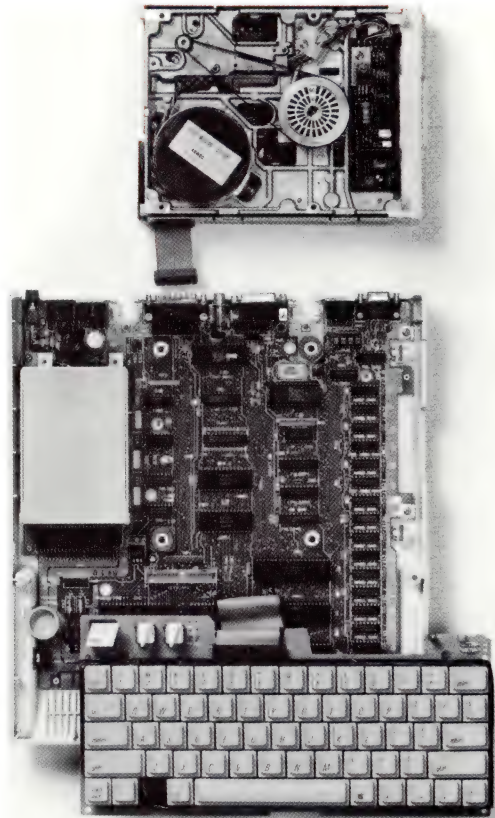
- The **memory management unit** (MMU) contains most of the logic that controls memory addressing in the Apple IIc.
- The **input/output unit** (IOU) contains most of the logic that controls the built-in input and output features of the Apple IIc.
- The **timing generator** (TMG) generates all the system and I/O clock and timing signals from a 14-MHz oscillator.
- The **general logic unit** (GLU) performs the remaining required logic functions.
- The **disk controller unit**, also known as the Integrated Woz Machine (IWM), is a single-chip version of the Apple Disk II controller card. It controls the built-in and external disk drives connected to the Apple IIc.

1.2.3 The Other Circuit Boards

The Apple IIc contains other circuit boards that serve special purposes: a motor-speed control and read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.

▲Warning

Adjustment of disk drive speed must be done by an authorized Apple Service Center. Do not attempt to adjust the speed of your built-in disk drive. If you do, you may damage it and you **will** void your warranty.



This chapter introduces the Apple IIc's processor, the 65C02, and the memory ranges and locations in the Apple IIc that have been set aside for special purposes. The last section of this chapter describes the reset routines, which restore the computer to a known state.

2.1 The 65C02 Microprocessor

The 65C02 is a general-purpose 8-bit CMOS microprocessor similar in operation to the 6502 used in other members of the Apple II family of computers.

Figure 2-1 is a model of the 65C02 microprocessor's register organization. Registers are fast-acting built-in storage areas where the processor performs and keeps track of its work. The 65C02 has one 16-bit register and five 8-bit registers.

Each of the other registers holds eight bits (one byte), so the 65C02 is called an **8-bit processor**.

The 16-bit register is called the program counter (PC). It specifies the address in memory that contains the instruction the processor is currently carrying out. A 16-bit register can specify any one of 65,536 memory addresses, and so the 65C02 is said to have an address space of 65,536 locations.

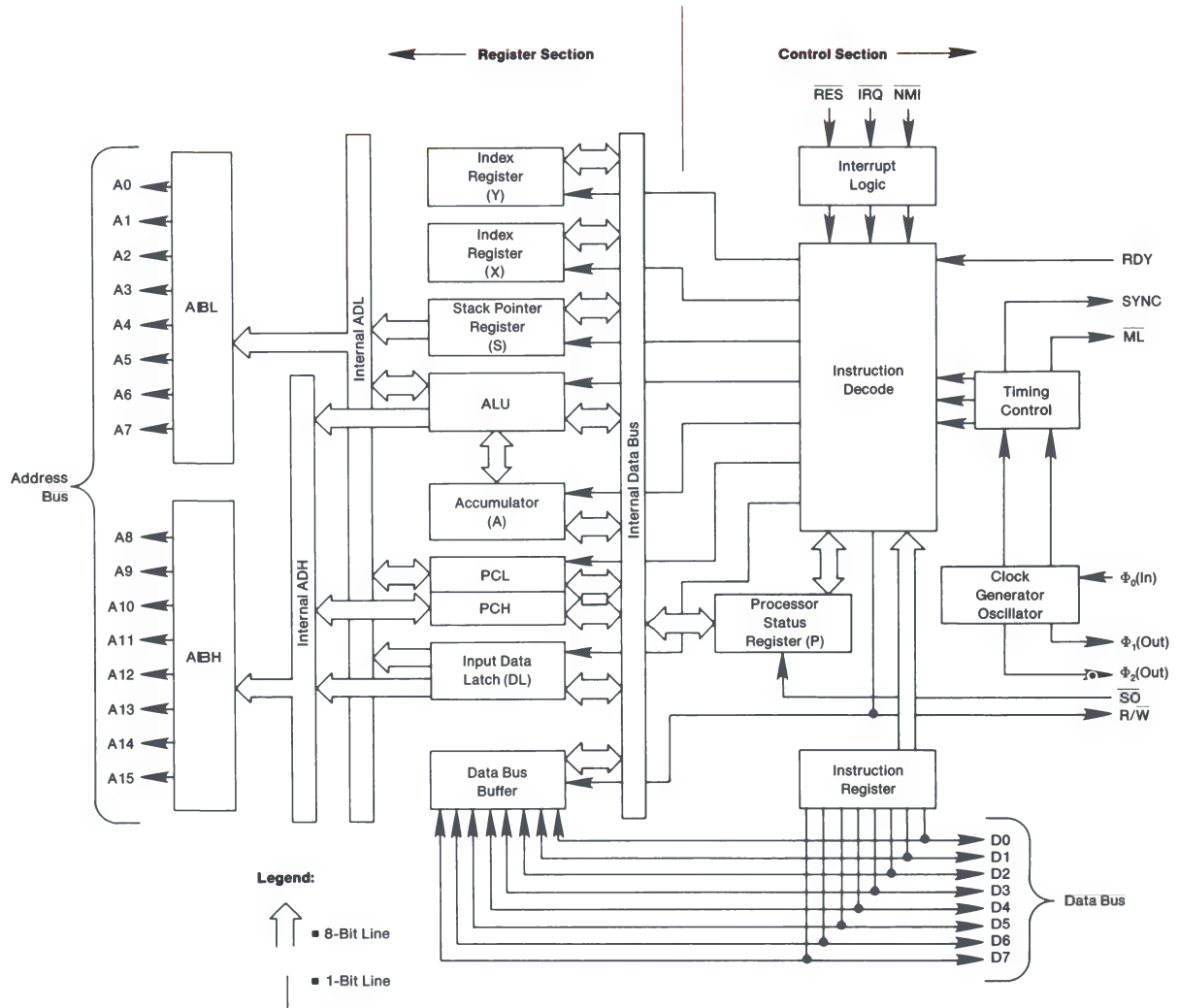
The five 8-bit registers in the 65C02 are the following:

- The **accumulator**, or A register. The accumulator is like a desk top where the processor performs mathematical and logical operations on information.
- The **index registers**, X and Y. The processor uses these registers to modify the address where information is to be found or placed, and to pass information from one program to another.
- A **stack pointer**, or S register. The processor uses a 256-byte region of memory—page \$01—as an area to stack up bytes for future use. The stack is empty when the computer is turned on. Several 65C02 instructions either push (store) the contents of a register onto the stack, or pull (retrieve) a byte from the stack and place it in a register. The S register keeps track of the address of the byte in the stack that is currently ready for use.
- A **processor status register**, called the P register. Seven of the eight bits of this register are used as flags to record the outcome of processor activities, and can be checked by later instructions to determine what has happened and what the processor should do next.

Appendix A lists the instructions the 65C02 can carry out, their use, and their effects on the registers. For further information, consult the pertinent books listed in the Bibliography.

Figure 2-1. Internal Model of the 65C02 Microprocessor

Copyright 1982, NCR Corporation. Used by permission of NCR Corporation, Dayton, Ohio.



2.2 Overview of the Address Space

Soft switches are described in Sections 2.4 and 2.5.

There are two other ROMs in the Apple IIc: one to generate characters corresponding to keystrokes (Section 11.7), and another to generate characters for display (Section 11.9). However, these ROMs are not addressable by the microprocessor.

The Apple IIc's 65C02 microprocessor can address 65,536 (64K) memory locations. All the Apple IIc's RAM, ROM, and input and output (I/O) devices are accessed using addresses in this 64K address range. Some functions have the same addresses—but not at the same time. The Apple IIc controls its shared addresses by using soft switches. A soft switch is a memory location that controls some aspect of the computer's operation when it is accessed.

All input and output in the Apple IIc is memory mapped—that is, specific memory addresses (all in the \$C0 page) are allocated to each I/O device. In this chapter, the I/O memory spaces are described simply as areas of memory. For details of the built-in I/O features and firmware, refer to the descriptions in Chapters 3 through 9.

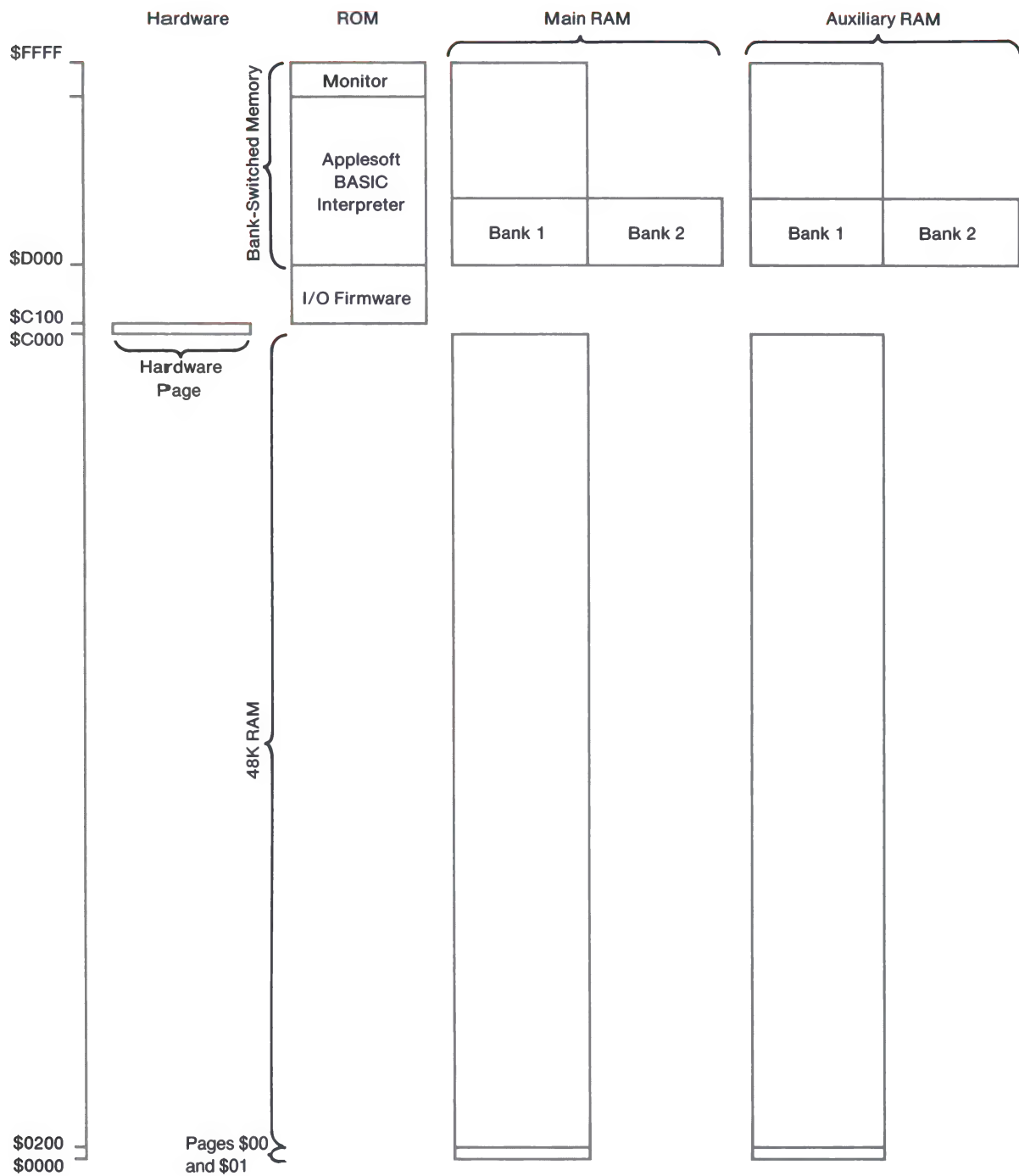
A contiguous block of 256 address locations in the 65C02's address range is called a page. A 1-byte address counter or 8-bit register can specify 1 of 256 different locations. Thus, page \$00 consists of memory locations from 0 through 255 (hexadecimal \$00 through \$FF); page \$01 consists of locations 256 through 511 (hexadecimal \$0100 through \$01FF); and so on. In this manual, all page numbers are given in hexadecimal format.

Note: The first two digits of a four-digit hexadecimal address are the page number. There are 256 pages of 256 bytes each in the address space. This kind of page is different from the display areas in the Apple IIc, which are sometimes referred to as Page 1 and Page 2.

2.3 Memory Map and Memory Switching

Figure 2-2 is a map of the Apple IIc's memory address space and what the major blocks of addresses are used for. As you can see in the figure, addresses \$C000 through \$C0FF contain hardware only, and addresses \$C100 through \$CFFF contain ROM only. At all other addresses there are two to five blocks of RAM or ROM locations. At any given time, only one block of RAM or ROM occupies each set of addresses. As described later in this chapter, soft switches in the hardware page control which blocks the processor is currently using.

Figure 2-2. Apple IIc Memory Map



2.3.1 Main RAM Addresses (\$0000–\$BFFF and \$D000–\$FFFF)

The area labeled *Main RAM* in Figure 2-2 is so-called because some or all of it is present in all models of the Apple II series of computers. The Apple IIc has 64K bytes of main RAM.

2.3.2 Auxiliary RAM Addresses (\$0000–\$BFFF and \$D000–\$FFFF)

The Apple IIc has 64K of auxiliary RAM built in. Some or all of that range of auxiliary memory is present in an Apple IIe with one of the 80-column text cards installed (see Appendix F), but there is no auxiliary RAM in the Apple II or II Plus.

A range of addresses in auxiliary RAM cannot be used simultaneously with the same range of addresses in main RAM; your programs must use the soft switches described in this chapter to select either main or auxiliary memory for any given range of addresses.

2.3.3 ROM Addresses (\$C100–\$FFFF)

ROM addresses contain the built-in Apple IIc firmware. Addresses \$C100 through \$CFFF belong exclusively to ROM. Addresses \$D000 through \$FFFF are shared by ROM, main RAM, and auxiliary RAM; the selection techniques are described in Section 2.4.2.

The Apple IIc's built-in ROM pages \$C1 through CF (addresses \$C100 through \$CFFF) contain I/O firmware. The Apple IIc I/O firmware is roughly divided among the built-in I/O devices as follows:

- Serial port 1 (RS-232 device) firmware entry points are on page \$C1. Much, but not all, of the firmware for the port is in the \$C100 space.
- Serial port 2 (communication device) firmware entry points are on page \$C2. Much, but not all, of the firmware for the port is in the \$C100 space.
- Video output firmware entry points are on page \$C3; the enhanced video firmware and miscellaneous I/O support routines occupy pages \$C8 through \$CF. This is partly because there are no slots 8 through F on the Apple IIc and because the firmware takes up more than one page of firmware memory space.

- Mouse firmware entry points are on page \$C4.
- Disk I/O firmware entry points are on page \$C6.

Note: This correspondence of ports and entry points does not imply that all of each port's firmware occupies a specific page. The Apple IIc I/O port firmware space is allocated in a way that provides the best possible performance in the available space.

The ROM address range of pages \$D0 through \$FF contain the Applesoft BASIC interpreter and the Monitor firmware, allocated as follows:

- Pages \$D0 through \$F7 (addresses \$D000 through \$F7FF) contain the Applesoft interpreter firmware.
- Pages \$F8 through \$FF (addresses \$F800 through \$FFFF) contain the Monitor, described in Chapter 10. You can use some of the built-in Monitor routines to make input and output procedures in your assembly-language programs easier to write. These routines are described in Chapters 3 through 9.

The operation of the Applesoft interpreter firmware is described in the *Applesoft BASIC Programmer's Reference Manual*.

2.3.4 Hardware Addresses (\$C000–\$C0FF)

Chapters 3 through 9 describe the Apple IIc's input and output locations. Appendix B lists all of these locations in address order, rather than by function.

The soft switches that the Apple IIc and your programs use to control the Apple IIc's built-in input and output functions are all found in the \$C0 memory page (addresses \$C000 through \$C0FF). In the same range of memory are the switches for selecting blocks of memory throughout the address space. This chapter describes the address space (memory) switches.

The hardware functions of the switches in this page fall into five basic categories:

- **Data inputs.** The only data input is location \$C000, where the low-order seven bits (bits 6 through 0) represent the keyboard key just pressed. (These data are guaranteed valid only when bit 7 = 1.)
- **Flag inputs.** Most built-in input locations are single-bit flags in the high-order (bit 7) position of their respective memory addresses. Flags have only two values: on (greater than or equal to 128 or \$80) or off (less than 128 or \$80).

The switch, hand controller (analog) and button inputs, and the keyboard strobe are examples of flag inputs. The locations for reading soft-switch states are also of this type.

Bit numbering in a byte is explained in Appendix H.

- **Strobe outputs.** The clear keyboard strobe (Chapter 4) and paddle timer strobe (Chapter 9) outputs are controlled by memory locations. If your program reads the contents of one of these locations, then the function associated with that location will be activated.
- **Toggle switches.** The Apple IIc has only one toggle switch: the speaker switch. A toggle switch has only one address assigned to it; each time you access it, it changes to its other state (on or off).

Reading the speaker toggle at location \$C030 clicks the speaker once. However, if you write to the speaker location, the microprocessor activates the address bus twice during successive clock cycles, causing the speaker toggle to end up in its original state before the speaker cone can move. Therefore, you should read, rather than write, to use this device.

The processor cannot read the on/off status of the speaker switch.

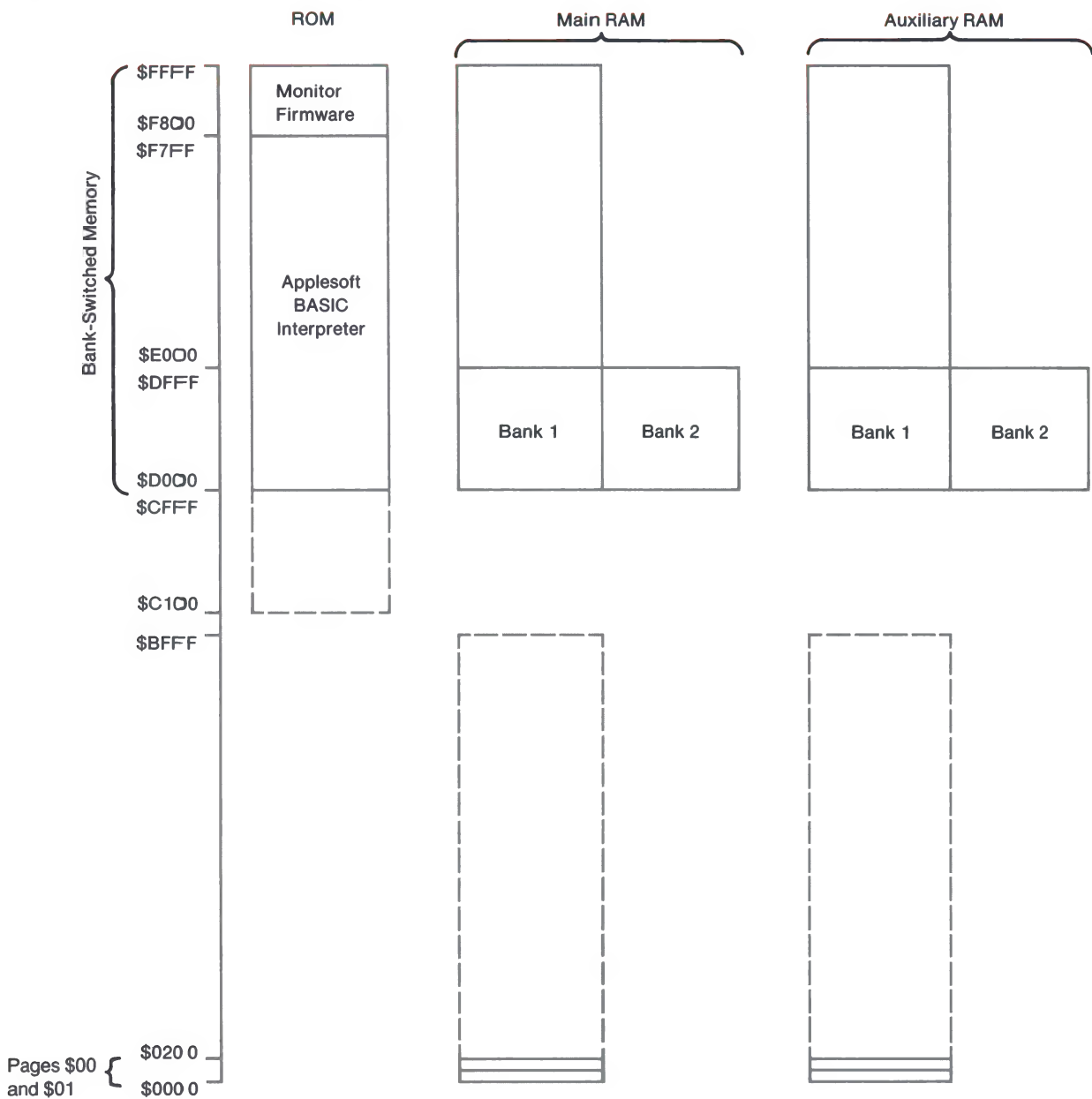
- **Soft switches.** Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them for reading the state of the switch.

There are eight soft switches that select different combinations of bank-switched memory (Section 2.4). Four of these eight switches require that your program read them twice in succession to activate them.

2.4 Bank-Switched Memory

The memory areas described in this section are called bank-switched memory (Figure 2-3) because so many banks (ranges) of addresses—one bank of ROM and up to four banks of RAM—occupy the same group of locations among the upper addresses of memory. Pages \$00 and \$01, at the low end of memory, are included here because the two sets of them—one in main RAM and one in auxiliary RAM—are controlled by the same switches as the high-address banks. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K memory space (Section 2.5).

Figure 2-3. Bank-Switched Memory Map



2.4.1 Page Allocations

Pages \$00 and \$01 are used by many of the 65C02 instructions. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

Page \$00 (One-Byte Addresses)

Several of the 65C02 microprocessor's addressing modes—for example, indirect addressing—require the use of addresses in page \$00, or zero page. However, the Monitor, the interpreters, and the operating systems all make extensive use of page \$00, too. One way to avoid conflicts is to use only those page-\$00 locations not already used by these other programs. But there is another way.

As you can see from Table B-1 in Appendix B, page \$00 is pretty well used up, except for a few bytes here and there. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: turn off interrupts, save the contents of part of page \$00, use that part, then restore the previous contents to page \$00, restore interrupts to their previous state, and then pass control to another program.

Page \$01 (The 65C02 Stack)

The 65C02 microprocessor uses page \$01 as its stack—a place where it can store subroutine return addresses, in last-in, first-out sequence. Programs can also use the stack for temporary storage of registers (via push and pull instructions). However, programs should use the stack carefully.

Pages \$D0–\$FF (ROM and RAM)

All these memory banks are controlled by the soft switches described in Section 2.4.2.

The memory address space from locations \$D000 through \$FFFF is used for both ROM and RAM. The 12K bytes of ROM in this address space contain the Monitor and the Applesoft BASIC interpreter.

There are 16K bytes of main RAM in this 12K space, with two banks occupying the 4K of addresses from \$D000 through \$DFFF. The RAM is normally used for storing other languages such as Pascal, or operating systems such as ProDOS™.

There are also 16K bytes of auxiliary RAM in this 12K space, again with double occupancy in the address range \$D000 through \$DFFF.

2.4.2 Using Bank Selector Switches

You switch banks of memory in the same way you switch other functions in the Apple IIc: by using soft switches. These soft switches do four things:

- Select either RAM or ROM in this memory space.
- Allow or inhibit (write-protect) writing to the RAM when RAM is selected.
- Select the first or second 4K-byte bank of RAM in the address space \$D000 through \$DFFF.
- Select either main RAM or auxiliary RAM.

▲Warning

Do not use soft switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program.

Table 2-1 shows the addresses of the soft switches for selecting all allowed combinations of reading and writing in this memory space, and the addresses of the locations to read the switch settings. Figures 2-4 through 2-10 illustrate how to select the combinations and what the resulting status of each switch is.

To make sure you do not inadvertently remove write protection from bank-switched RAM, the four write-enable addresses require that you read them twice in succession (indicated by RR in Table 2-1).

Because the AltZP switch shares the read keyboard address, you must write (W in Table 2-1) to its locations to change the switch setting.

To find out which way a switch is set, read the appropriate location and then check bit 7 (shown as R7 in Table 2-1). If the bit is a 1, the answer to the question given in the table is affirmative.

Note that there is no way to check whether write protection is on or off.

Important!

You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-5 and 2-6), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program there.

Table 2-1. Bank Selector Switches

Name	Action	Hex	Dec	Function
	R	\$C080	49280	Read RAM; no write; use \$D000 bank 2
	RR	\$C081	49281	Read ROM; write RAM; use \$D000 bank 2
	R	\$C082	49282	Read ROM; no write; use \$D000 bank 2
	RR	\$C083	49283	Read and write RAM; use \$D000 bank 2
	R	\$C088	49288	Read RAM; no write; use \$D000 bank 1
	RR	\$C089	49289	Read ROM; write RAM; use \$D000 bank 1
	R	\$C08A	49290	Read ROM; no write; use \$D000 bank 1
	RR	\$C08B	49291	Read and write RAM; use \$D000 bank 1
RdBnk2	R7	\$C011	49169	Read whether \$D000 bank 2 (1) or bank 1 (0)
RdLCRAM	R7	\$C012	49170	Read RAM (1) or ROM (0)
AltZP	W	\$C008	49160	Off: Use main bank, page \$00 and page \$01
AltZP	W	\$C009	49161	On: Use auxiliary bank, page \$00 and page \$01
RdAltZP	R7	\$C016	49174	Read whether auxiliary (1) or main (0) bank

Figure 2-4. Read ROM

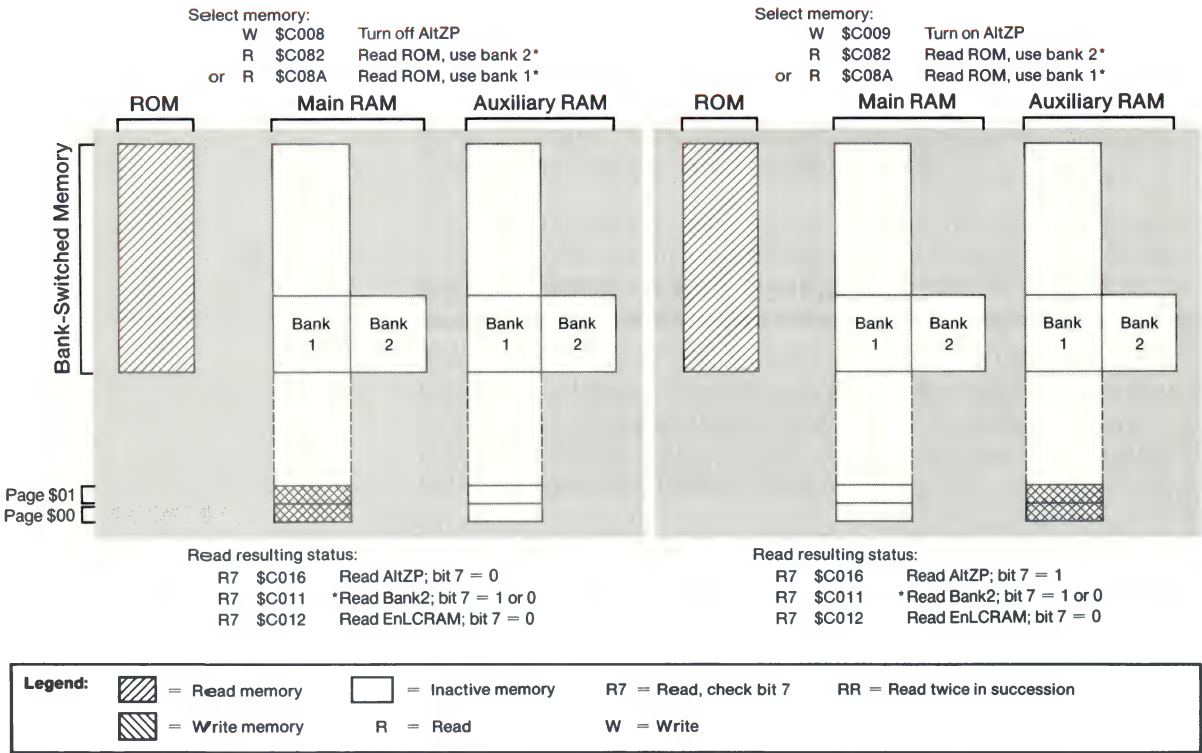


Figure 2-5. Read ROM, Write RAM, and Use First \$D0 Bank

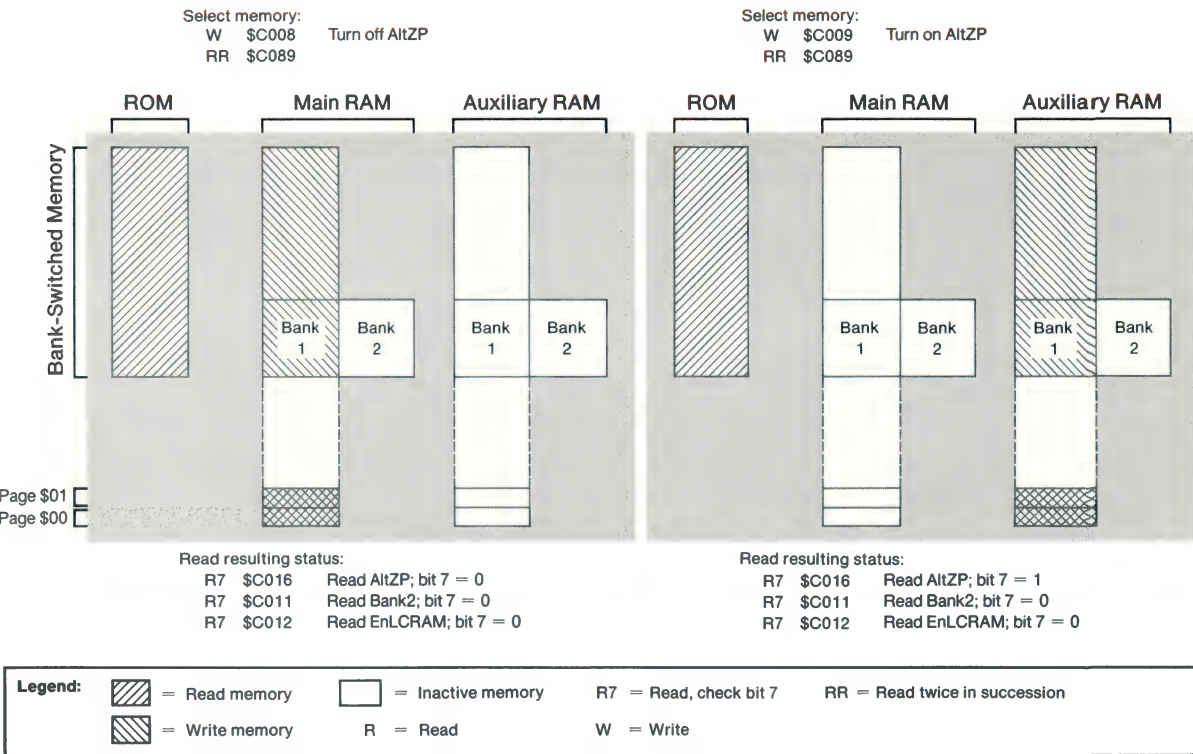


Figure 2-6. Read ROM, Write RAM, and Use Second \$D0 Bank

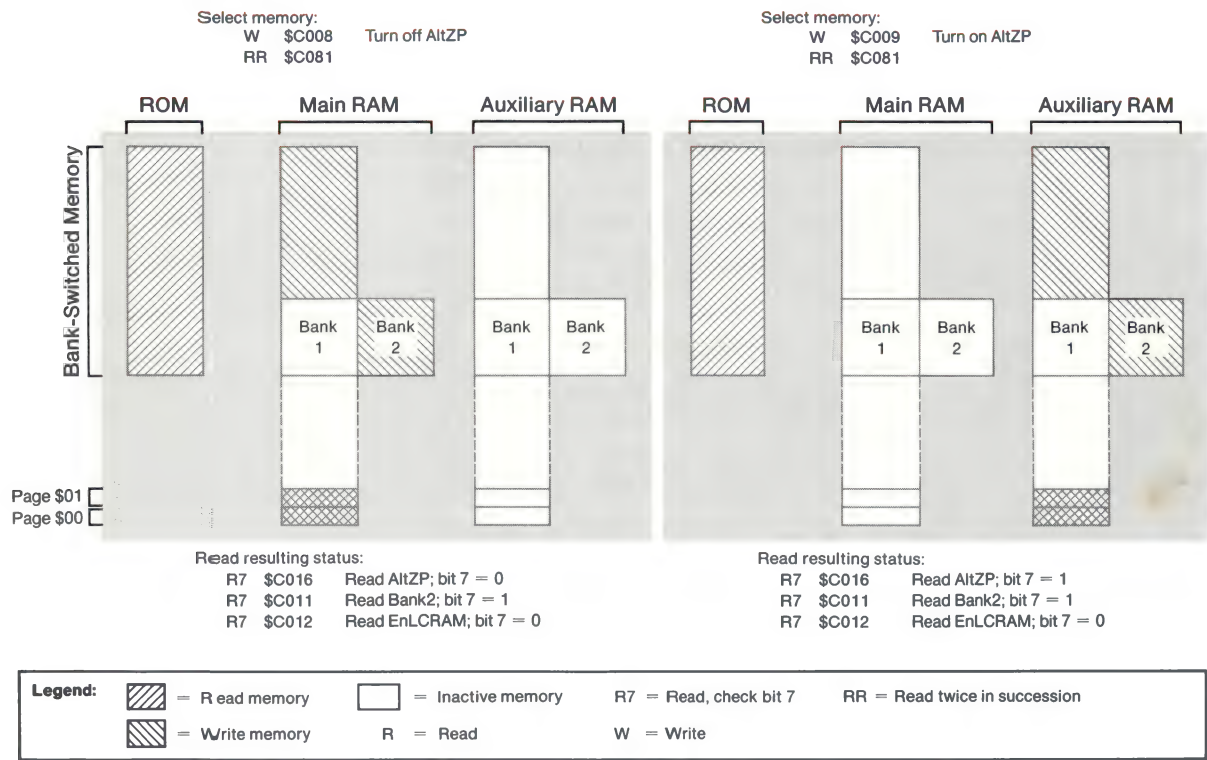


Figure 2-7. Read RAM and Use First \$D0 Bank

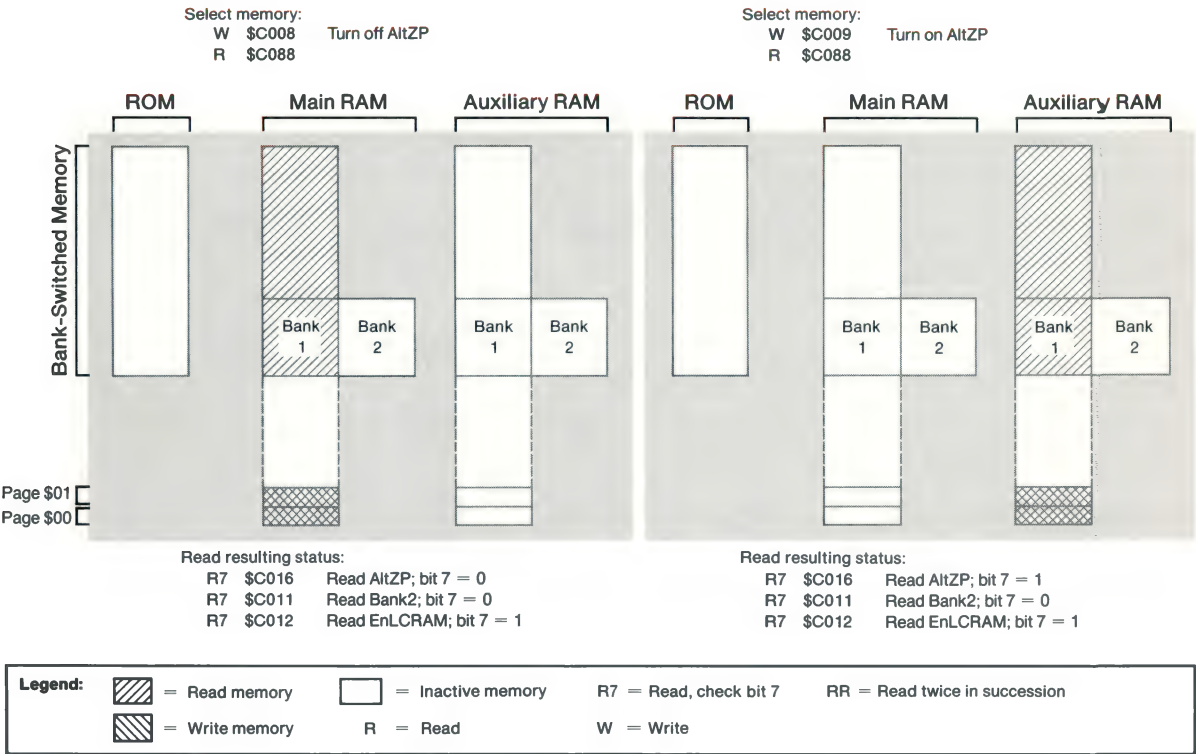


Figure 2-8. Read RAM and Use Second \$D0 Bank

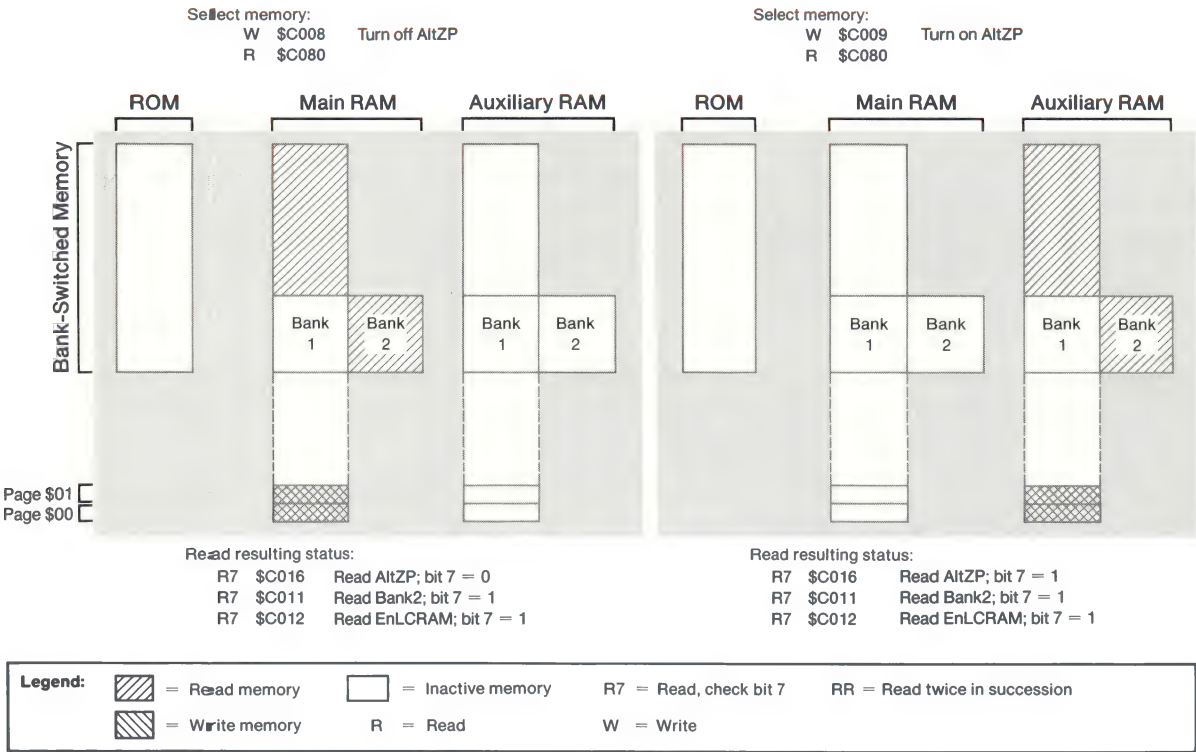


Figure 2-9. Read and Write RAM and Use First \$D0 Bank

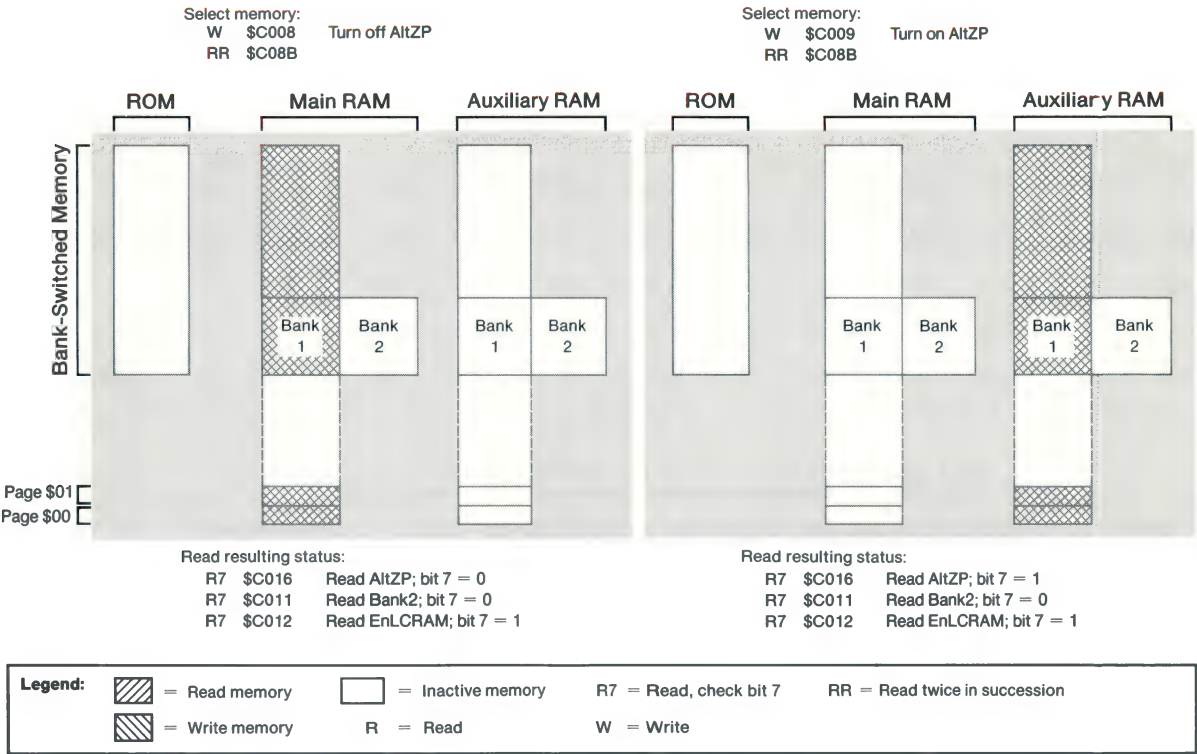
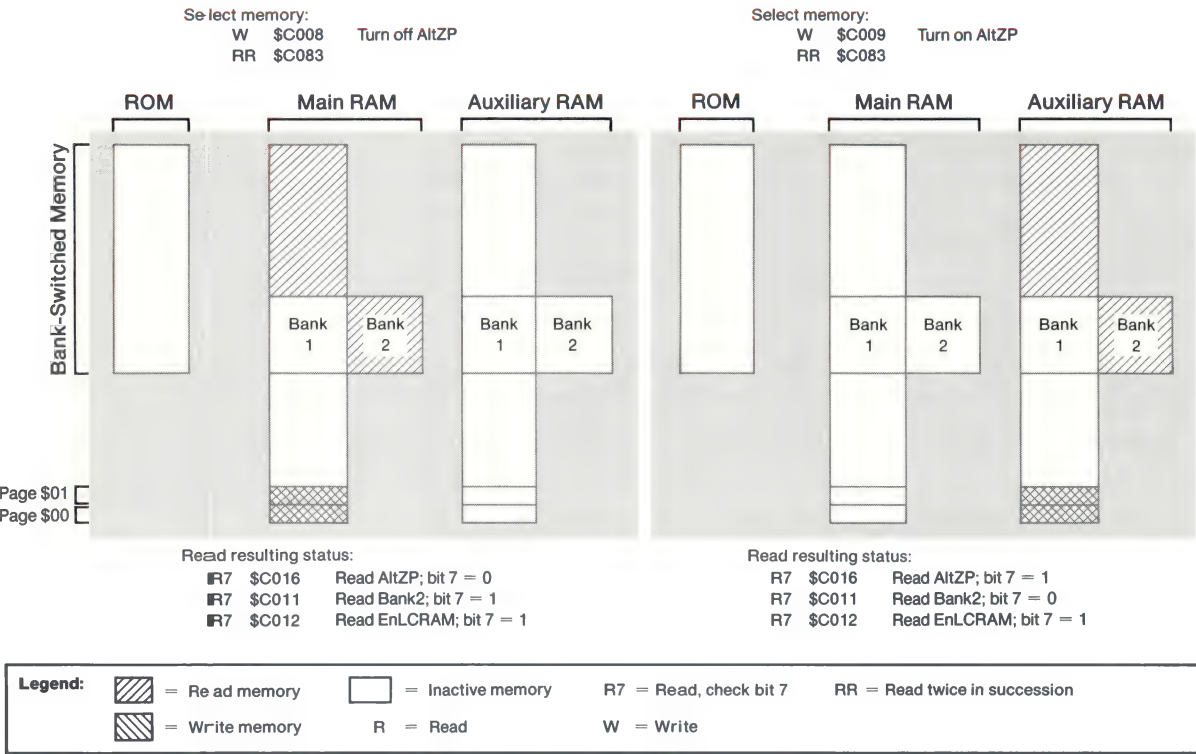


Figure 2-10. Read and Write RAM and Use Second \$D0 Bank



2.5 48K Memory

The 48K memory space (actually, 47½K) extends from location \$0200 to location \$BFFF (Figure 2-11) in both main and auxiliary RAM. The amount of storage available in this address space depends on what language or operating system you are using, and what video display needs your program has.

2.5.1 Page Allocations

Most of the Apple IIc's 48K RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware, the Applesoft BASIC interpreter, and whatever video display you may select.

Important!

The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain.

A **buffer** is any storage area set aside for one program or device to put information into and another to take information out of at a different time or rate.

Page \$02 (The Input Buffer)

The GetLn input routine (Section 3.2.3) uses page \$02 as its keyboard-input buffer. The size of this buffer (256 bytes) sets the maximum size of input strings read by Applesoft or the Monitor. If you know that you won't be typing any long input strings (more than, say, 30 characters), you can store temporary data at the upper end of page \$02.

Refer to Appendix D and to the appropriate programmer and reference manuals for operating system use of page \$03.

Page \$03 (Global Storage and Vectors)

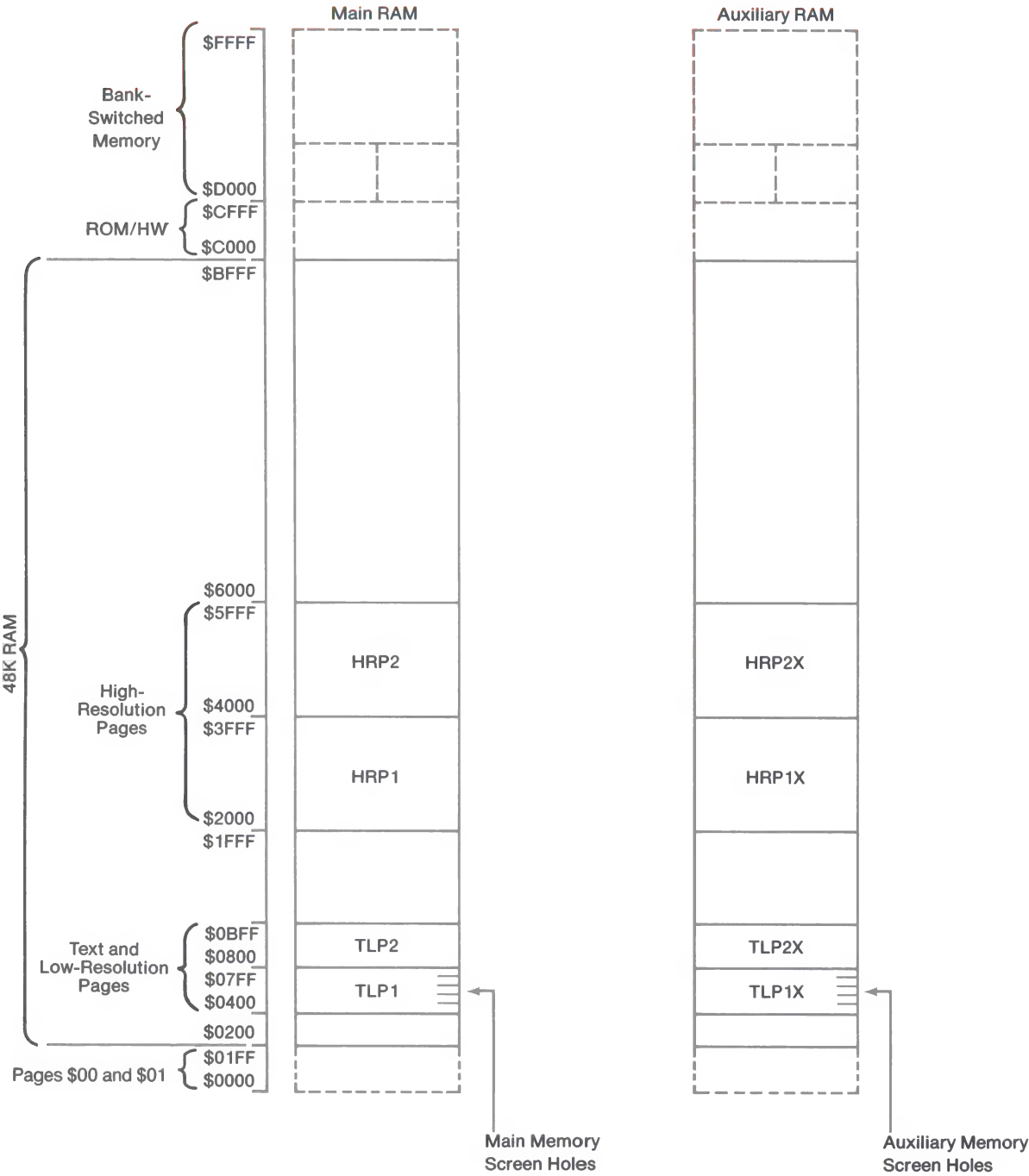
The Monitor and operating systems use parts of page \$03 for global storage and vectors. Table 2-7, later in this chapter, shows the part of page \$03 the built-in firmware uses.

Global storage refers to an area reserved for information that programs use in common. **Vectors**—the addresses of special routines—are examples of this kind of information. Section 2.6 discusses the global storage and vectors found on page \$03.

Pages \$04–\$07 (Text and Low-Resolution Page 1)

The most often used display buffer is the text and low-resolution graphics Page 1 (TLP1 in Figure 2-11), which occupies main memory pages \$04 through \$07. It is not usable for program and data storage if you are using Monitor routines or Applesoft, or with almost any other program that uses text or low-resolution display.

Figure 2-11. 48K Memory Map



See Chapter 5.

Text and low-resolution Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages \$04 through \$07. This pair of text and low-resolution graphics pages are used together to produce 80-column text display.

See Section 3.4.6.

There are 128 locations in pages \$04 through \$07 (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called screen holes.

▲Warning | The screen holes are reserved for use by the built-in firmware.

Pages \$08–\$0B (Text and Low-Resolution Page 2)

The second text and low-resolution graphics display buffer, TLP2, occupies main memory pages \$08 through \$0B. Most programs do not use Page 2 for displays, but TLP2 is there for display use if required.

Text and low-resolution Page 2X (TLP2X) is an identical display buffer occupying pages \$08 through \$0B in auxiliary memory.

Note that Apple IIc firmware does not provide a way to use the second pair of text and low-resolution graphics pages for 80-column text display.

Page \$08 (Communication Port Buffers)

Serial port 2: See Chapter 8.

Serial port 2 uses the first half of auxiliary memory page \$08 (addresses \$0800 through \$087F) as a keyboard input buffer, and the second half of the page (addresses \$0880 through \$08FF) as a serial input buffer. These buffers increase the data transfer rates possible with the serial communication port. Appendix E explains how to use these features.

If your program does not use this page for buffers, it can use it as part of TLP2X.

Pages \$20–\$3F (High-Resolution Page 1)

The primary high-resolution graphics display buffer, called high-resolution Page 1 (HRP1), occupies the 32 memory pages from \$20 through \$3F (locations \$2000 through \$3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution Page 1X (HRP1X) is an identical display page occupying auxiliary memory pages \$20 through \$3F.

See Chapter 5.

The Apple IIc can display double-high-resolution graphics by interleaving HRP1 and HRP1X.

Pages \$40–\$5F (High-Resolution Page 2)

High-resolution Page 2 occupies main memory pages \$40 through \$5F (locations \$4000 through \$5FFF). Most programs use this area for program or data storage, but it is also available as a second high-resolution page.

High-resolution Page 2X (HRP2X) occupies auxiliary memory pages \$40 through \$5F.

For more information about the display buffers, see Chapter 5.

Apple IIc firmware provides high-resolution graphics routines for HRP1 and HRP2 only. Refer to the *Applesoft BASIC Programmer's Reference Manual*.

2.5.2 Using 48K Memory Switches

Two switches select main or auxiliary RAM in the 48K memory space: RAMRd determines which to use for reading, and RAMWrt determines which to use for writing. When these switches are on, they select auxiliary memory. When they are off, they select main memory. (This discussion assumes that the 80Store switch, used to control display memory, is off.)

For details, refer to Section 2.5.4.

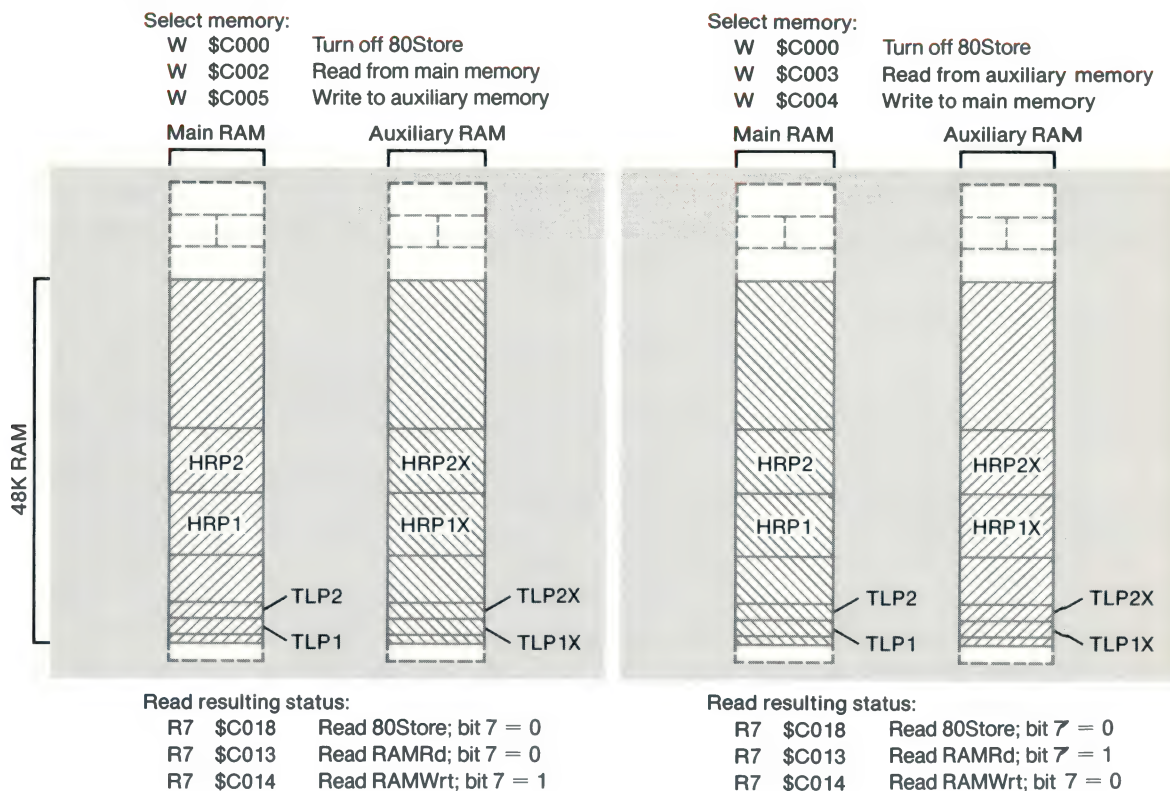
Each switch has three locations assigned to it (Table 2-2): one to turn it on, one to turn it off, and a third to read its state. Because the memory locations for turning the switches on and off are shared with keyboard reading functions, you must write to these addresses to use them for memory switching. For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0. Figures 2-12 and 2-13 illustrate how the switches work.

Table 2-2. 48K Memory Switches

Note: 80Store must be off to switch all memory in this range, including display memory (Table 2-6).

Name	Action	Hex	Dec	Function
RAMRd	W	\$C002	49154	Off: Read main 48K RAM
RAMRd	W	\$C003	49155	On: Read auxiliary 48K RAM
RdRAMRd	R7	\$C013	49171	Read whether main (0) or aux. (1)
RAMWrt	W	\$C004	49156	Off: Write to main 48K RAM
RAMWrt	W	\$C005	49157	On: Write to auxiliary 48K RAM
RdRAMWrt	R7	\$C014	49172	Read whether main (0) or aux. (1)

Figure 2-12. 48K RAM Selection: Split Pairs



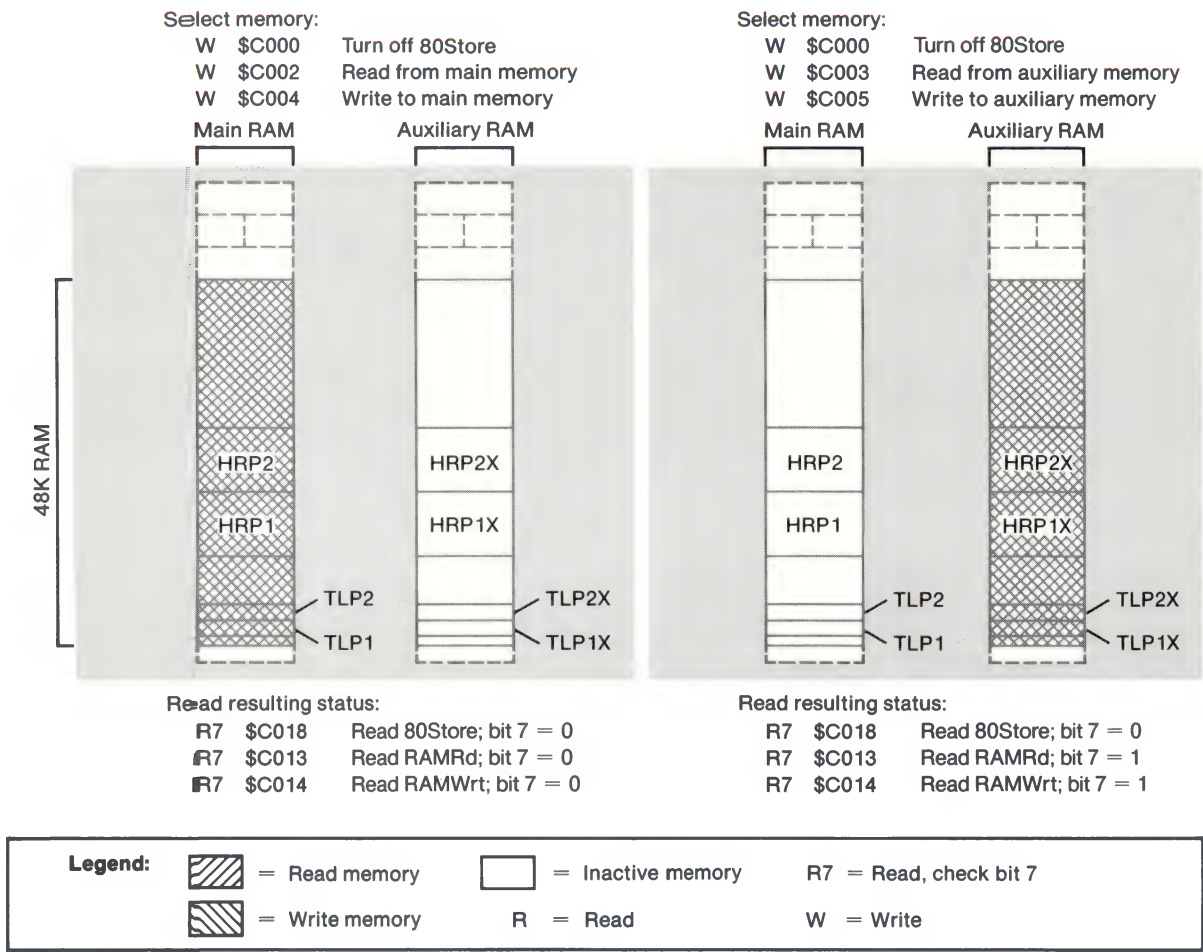
Legend:

= Read memory
 = Write memory

= Inactive memory
R = Read

R7 = Read, check bit 7
W = Write

Figure 2-13. 48K RAM Selection: One Side Only



2.5.3 Transfers Between Main and Auxiliary Memory

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in 48K RAM transfer routines. These routines (listed in Table 2-3) make it possible to move between main and auxiliary memory without having to manipulate the soft switches described in Section 2.5.2.

Important! The routines described below make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program.

Table 2-3. 48K RAM Transfer Routines

Name	Action	Hex	Function
MoveAux	JSR	\$C311	Move data blocks between main and auxiliary 48K memory
XFer	JMP	\$C314	Transfer program control between main and auxiliary 48K memory

Transferring Data

In your assembly-language programs, you can use the built-in routine named MoveAux to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page \$00 and set or clear the carry bit to select the direction of the move.

Warning Don't try to use MoveAux to copy data in bank-switched memory (page \$00, page \$01, or pages \$D0 through \$FF). MoveAux uses page \$00 all during the copy.

The pairs of bytes you use for passing addresses to this routine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIc's built-in routines. The addresses of these byte pairs are shown in Table 2-4.

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

Table 2-4. Parameters for MoveAux Routine

Name	Location	Parameter Passed
Carry		1 = Move from main to auxiliary memory 0 = Move from auxiliary to main memory
A1L	\$3C	Source starting address, low-order byte
A1H	\$3D	Source starting address, high-order byte
A2L	\$3E	Source ending address, low-order byte
A2H	\$3F	Source ending address, high-order byte
A4L	\$42	Destination starting address, low-order byte
A4H	\$43	Destination starting address, high-order byte
	X, Y, A	These registers are preserved.

The MoveAux routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit (SEC instruction); to copy data from auxiliary memory to main memory, clear the carry bit (CLC instruction).

When you make the subroutine call to MoveAux, the subroutine copies the block of data as specified by the A register and the carry bit. When it is finished, the accumulator and the X and Y registers are just as they were when you called it.

Transferring Control

You can use the built-in routine named XFer to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFer: the address of the routine you are transferring to, the direction of the transfer, and which page \$00 and stack you want to use (Table 2-5).

Table 2-5. Parameters for XFer Routine

Name	Location	Parameter Passed
Carry		1 = Transfer from main to auxiliary memory 0 = Transfer from auxiliary to main memory
Overflow		1 = Use page \$00 and stack in auxiliary memory 0 = Use page \$00 and stack in main memory
	\$03ED	Program starting address, low-order byte
	\$03EE	Program starting address, high-order byte
	X, Y, A	These registers are preserved.

Put the transfer address into the two bytes at locations \$03ED and \$03EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory.

Use the overflow bit to select which page \$00 and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit (cause an overflow condition) to use the auxiliary memory.

After you have set up the parameters, pass control to the XFer routine by a jump instruction, rather than a subroutine call.

▲Warning

It is your responsibility as the programmer to save the current stack pointer before using XFer and to restore it after regaining control. Failure to do so will cause program errors. Refer to Appendix E for instructions on how to do this.

2.5.4 Using Display Memory Switches

Section 2.5.2 discusses how to select main or auxiliary RAM for the 48K memory space. However, under many circumstances your program may want to control reading and writing to display pages separately. The switches discussed in this section override the effects of RAMRd and RAMWrt for display pages only.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (Table 2-6). One of the switches, 80Store, shares its on and off addresses with a keyboard reading function. As a result, your program must write to these locations to turn the switch on and off.

Table 2-6. Display Memory Switches

Name	Action	Hex	Dec	Function
80Store	W	\$C000	49152	Off: RAMRd and RAMWrt determine RAM locations.
80Store	W	\$C001	49153	On: Page2 switches between TLP1 and TLP1X, and (if HiRes on) between HRP1 and HRP1X.
Rd80Store	R7	\$C018	49176	Read whether 80Store on (1) or off (0)
Page2	R	\$C054	49236	Off: Select TLP1 and HRP1
Page2	R	\$C055	49237	On: If 80Store off, switch to TLP2, and (if HiRes on) to HRP2. If 80Store on, switch to TLP1X, and (if HiRes on) to HRP1X.
RdPage2	R7	\$C01C	49180	Read whether Page2 on (1) or off (0)
HiRes	R	\$C056	49238	Off: Display text and low-resolution page
HiRes	R	\$C057	49239	On: Display high-resolution pages; make Page2 switch between high-resolution pages
RdHiRes	R7	\$C01D	49181	Read whether HiRes on (1) or off (0)

See Chapter 5 for a discussion of display pages.

Table 2-6—continued. Display Memory Switches

Name	Action	Hex	Dec	Function
IOUDis	W	\$C07E	49278	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch*
IOUDis	W	\$C07F	49279	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*
RdIOUDis	R7	\$C07E	49278	Read IOUDis switch (1=off)†
DHiRes	R/W	\$C05E	49246	On: (If IOUDis on) turn on double-high resolution
DHiRes	R/W	\$C05F	49247	Off: (If IOUDis on) turn off double-high resolution
RdDHiRes	R7	\$C07F	49279	Read DHiRes switch (1 = on)†

* The firmware normally leaves IOUDis on.

† Reading or writing any address in the range \$C070–\$C07F also triggers the paddle timer and resets VBIInt (see Chapter 9).

For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

Here is how these switches work for reading and writing:

- If HiRes is off, then Page2 switches between text and low-resolution graphics pages (TLP) only. If HiRes is on, then Page2 switches between TLP and high-resolution graphics pages (HRP).
- If 80Store is off, RAMRd and RAMWrt (Table 2-2) determine whether main or auxiliary RAM locations are used. Page2 selects pages for display (Chapter 5), but not for reading and writing.
- If 80Store is on, it overrides RAMRd and RAMWrt with respect to the display pages selected by HiRes and Page2 (Figures 2-14 and 2-15).

Figure 2-14. Page2 Selections With 80Store On and HiRes Off

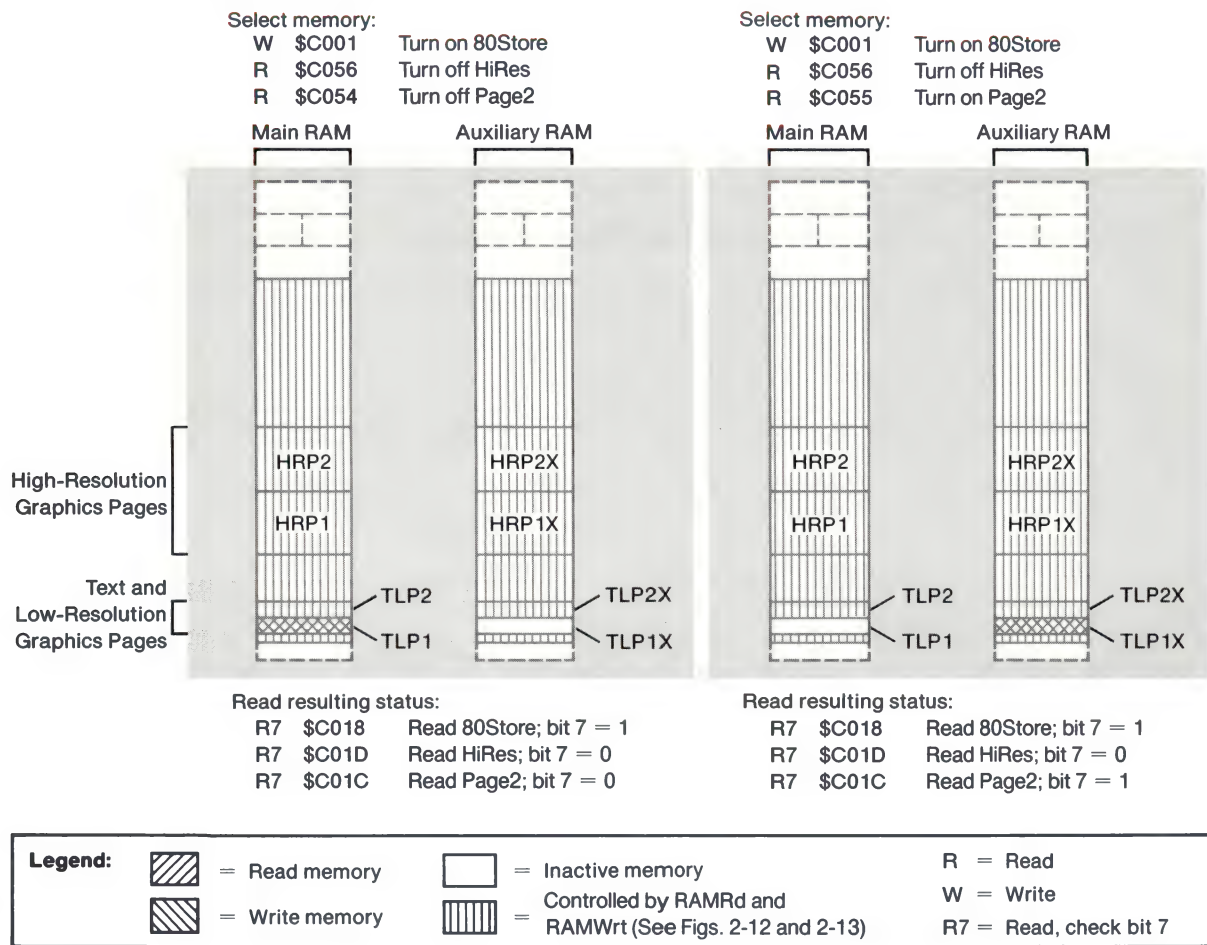
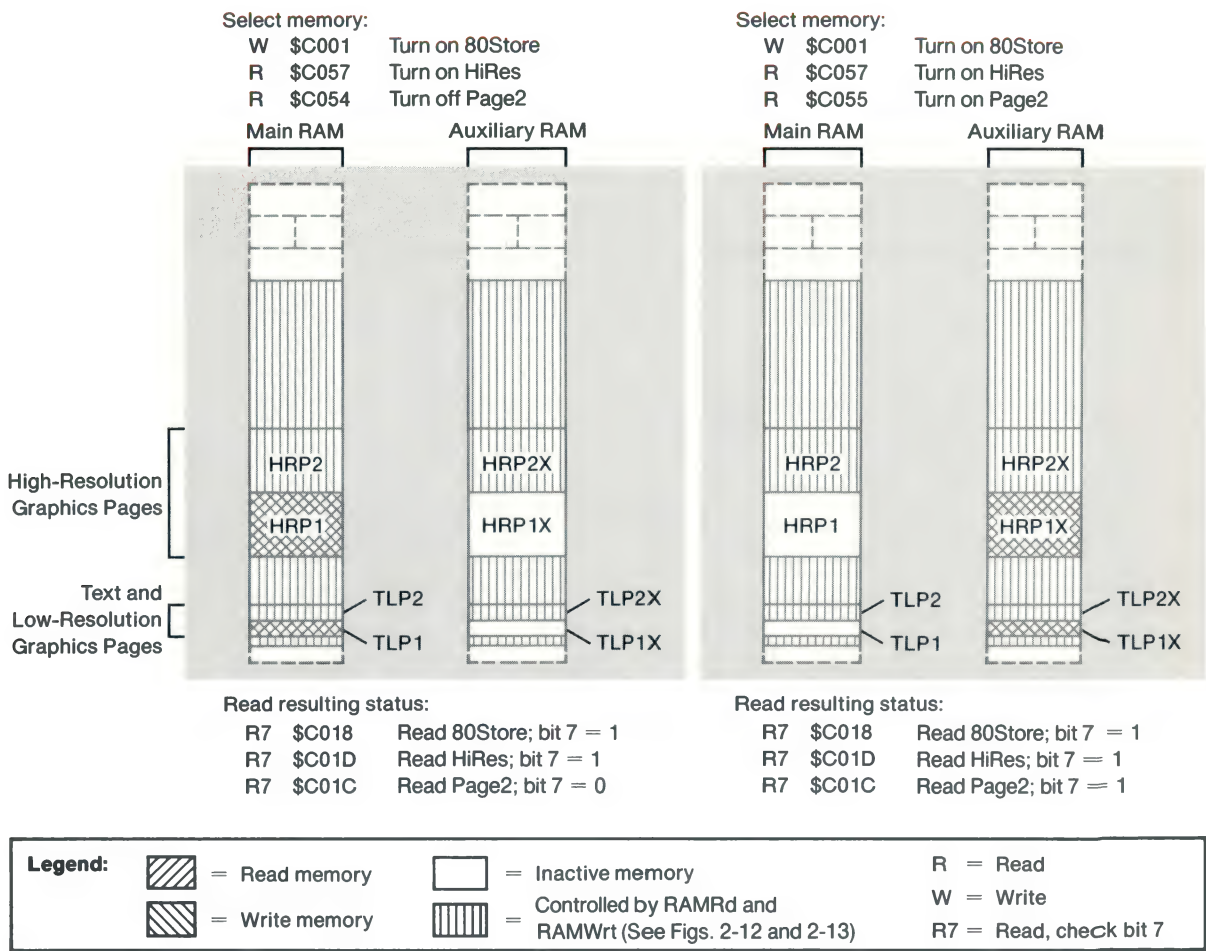


Figure 2-15. Page2 Selections With 80Store On and HiRes On



2.6 The Reset Routine

A procedure called the reset routine (Figure 2-16) puts the Apple IIc into a known state when it has just been turned on or when you hold down **CONTROL** while pressing **RESET**. The reset routine puts the Apple IIc into its normal operating mode and restarts the program indicated at locations \$03F2 and \$03F3 (Table 2-7).

When you initiate a reset, hardware in the Apple IIc sets the memory-controlling soft switches to normal: main ROM and RAM are enabled, auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at \$D000.

Figure 2-16. Reset Routine Flowchart

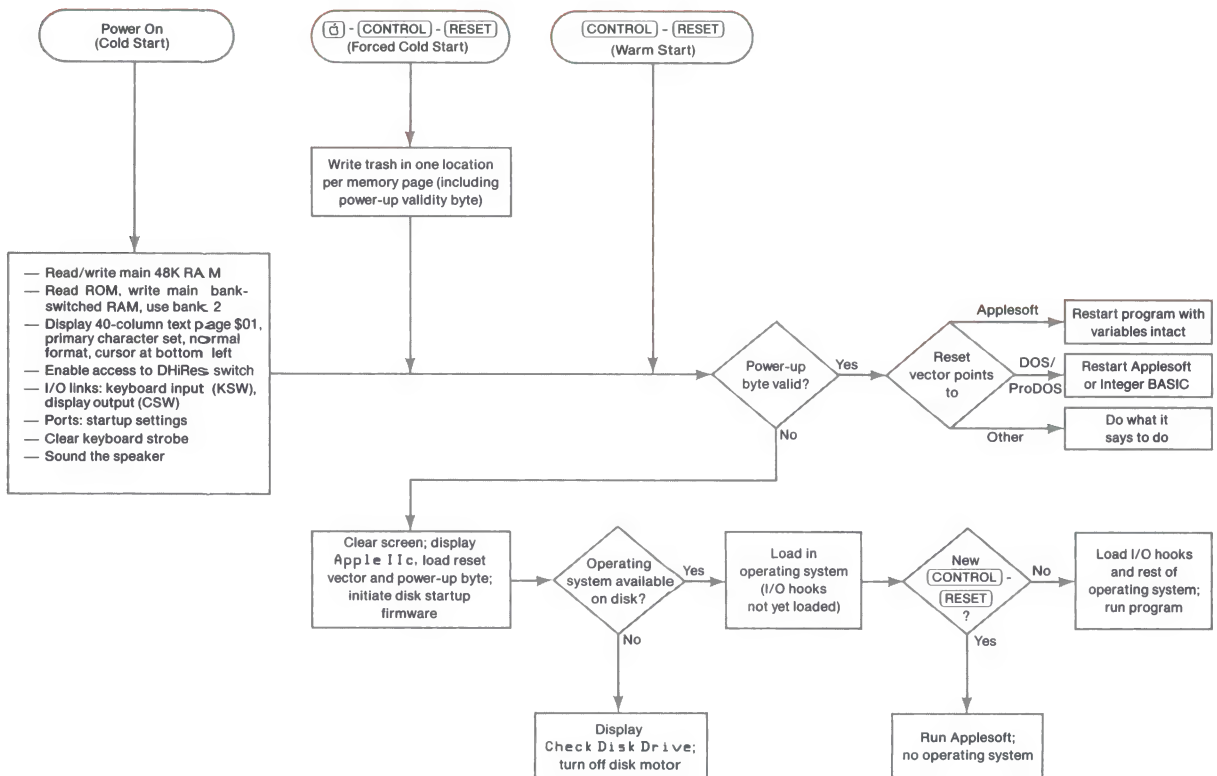


Table 2-7. Page \$03 Vectors

Vector Address	Vector Function
\$03F0 (1008)	Address of the subroutine that handles BRK requests (normally \$59, \$FA)
\$03F1 (1009)	
\$03F2 (1010)	Reset vector (see text)
\$03F3 (1011)	
\$03F4 (1012)	Power-up byte (see text)
\$03F5 (1013)	Jump instruction to the subroutine that handles Applesoft &-commands (normally \$4C, \$58, \$FF)
\$03F6 (1014)	
\$03F7 (1015)	
\$03F8 (1016)	
\$03F9 (1017)	Jump instruction to the subroutine that handles user CONTROL-Y commands
\$03FA (1018)	
\$03FB (1019)	Jump instruction to the subroutine that handles nonmaskable interrupts (not used on Apple IIc)
\$03FC (1020)	
\$03FD (1021)	
\$03FE (1022)	Interrupt vector (address of the subroutine that handles interrupt requests) (Appendix E)
\$03FF (1023)	

The reset routine sets the display-controlling soft switches to display 40-column text Page 1 using the primary character set, then sets the display window equal to the full 40-column display, puts the cursor at the bottom of the screen, and sets the text display format to normal.

The reset routine also sets the keyboard and display as the standard input and output devices (Chapter 3). It masks mouse interrupts and sets mouse defaults (Table 9-1). Finally, it enables DHiRes switch access (by turning on IOUDis), clears the keyboard strobe, and sounds the speaker.

The Apple IIc has three types of reset: power-on reset, also called cold-start reset; warm-start reset; and forced cold-start reset. The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is valid or not. If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

The reset vector validity check is described in Section 2.6.4.

2.6.1 The Cold-Start Procedure (Power On)

If the reset vector is not valid, either the Apple IIc has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string **Apple© IIc** at the top of the display. It loads the reset vector and the validity-check byte, then initiates the startup routine that resides in the disk controller firmware. The bootstrap routine then loads whatever operating system resides on the disk in the built-in drive. When the operating system has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor keeps spinning for a brief time. Then the firmware shuts it off and displays the message **Check Disk Drive** at the bottom of the screen.

If you press **CONTROL-RESET** again before the startup procedure is completed, the reset routine continues without using the disk, and passes control to the Applesoft BASIC interpreter.

2.6.2 The Warm-Start Procedure (CONTROL-RESET)

Whenever you press **CONTROL-RESET** when the Apple IIc has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the program it points to, which at power-up is the Applesoft interpreter.

If the vector does point to the Applesoft interpreter, your Applesoft program and variables are still intact. If you are using DOS or ProDOS, that operating system is the resident program and it restarts the BASIC interpreter you were using when you pressed **CONTROL-RESET**.


Important!

A program residing only in bank-switched RAM cannot use the reset vector to regain control after a reset, because upon reset the hardware selects the ROM for reading in the bank-switched memory space.



2.6.3 Forced Cold Start (OPEN APPLE-CONTROL-RESET)

If a program has set the reset vector to point to its own warm-start address, as described below, pressing **CONTROL-RESET** causes transfer of control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down **CONTROL** and **⌘**, then pressing and releasing **RESET**.



Important!

When you want to stop a program unconditionally—for example, to start up the Apple IIc with some other program—you should use the forced cold-start reset, -**CONTROL**-**RESET**, instead of turning the power off and on.

UniDisk 3.5

You must hold  down until the built-in drive starts to spin. If you release  before the drive starts to spin, the Apple IIc drops in to BASIC instead of rebooting.

The forced cold-start reset works as follows. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page \$03 are the ones that contain the reset vector. The warm-start reset routine finds the error, and so performs a normal cold-start reset.

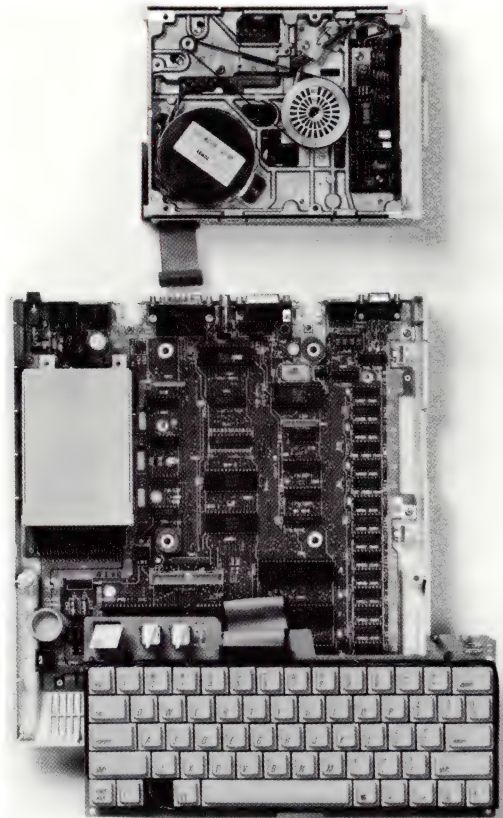
Note that if you press both  and  during power-up or **CONTROL**-**RESET**, built-in exercise code is executed. This code is for production and has no end-user value.

2.6.4 The Reset Vector

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations \$03F2 and \$03F3. It then stores a validity-check byte, also called the power-up byte, at location \$03F4. The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal \$A5). Each time you reset the Apple IIc, the reset routine uses this byte to determine whether the reset vector is still valid.

You can change the reset vector so that the reset routine will transfer control to your program instead of to the Applesoft interpreter. For this to work, you must also change the validity-check byte to the exclusive-OR of the high-order byte of your new reset vector with the constant 165 (\$A5). If you fail to do this, then the next time you reset the Apple IIc, the reset routine will determine that the reset vector is invalid and perform a cold-start reset, eventually transferring control to the disk bootstrap routine or to Applesoft.

There is a subroutine that generates the validity-check byte for the current reset vector. This subroutine, called SetPWRC, is at location \$FB6F. When your program finishes, it can return the Apple IIc to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.



This chapter is an introduction to the built-in I/O capabilities of the Apple IIc. It outlines

- standard I/O links and their functions
- I/O firmware protocols
- dedicated memory storage locations
- direct I/O.

The next six chapters discuss these capabilities in detail.

3.1 The Standard I/O Links

You can use some of the routines in the Apple IIc's firmware for your own programs. This can save you both program space and the time and effort of writing all your own I/O routines.

To use the built-in firmware routines, your program must perform a JSR to the routine's entry address. The called routine then performs an indirect jump through an address stored somewhere in RAM and begins executing. When the routine has finished doing its work, it returns (with an RTS) to your program at the first instruction following the JSR used to call the routine. Memory locations used for transferring control to other subroutines, such as the indirect jump's address used by the character I/O routine, are sometimes called vectors. In this manual, the locations used for transferring control to the Apple IIc's I/O subroutines are called the I/O links.

In an Apple IIc running without an operating system, each I/O link normally contains the address of the standard input or output subroutine. An operating system will typically place addresses of its own I/O routines in these link locations instead.

By calling the I/O subroutines that then jump to the routines pointed to by the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly with other software, such as the operating system or a device driver. The I/O links contain the addresses of KeyIn and COut1 if the enhanced video firmware is off (when the display shows a flashing checkerboard cursor), and of C3KeyIn and C3COut1 if that firmware is on (when the display shows an inverse solid cursor).

The standard I/O links are two pairs of locations in the Apple IIc RAM in the range \$36 through \$39 that are used for controlling character input and output.

Note: Not all operating systems use the standard I/O links. For example, Apple Pascal does not use them.

The link at locations \$36 and \$37 is called CSW, for character output switch. Individually, location \$36 is called CSWL (CSW low) and location \$37 is called CSWH (CSW high). This link holds the starting address of the subroutine the Apple IIc is currently using for single-character output. This address is normally \$FDF0, the address of routine COut1.

When you issue either a PR#n from BASIC or an n CONTROL-P from the Monitor, the Apple IIc changes this link address to the first address in the ROM space allocated to port n. That address has the form \$Cn00. Subsequent calls for character output are thus transferred to the firmware starting at that address. When it has finished, the firmware executes an RTS (return from subroutine) instruction to return control to the calling program. Sometimes a PR#n will cause both input and output switches to be changed (as in the 80-column firmware).

A similar link at locations \$38 and \$39 is called KSW, for keyboard input switch. Individually, location \$38 is called KSWL (for KSW low) and location \$39 is called KSWH (KSW high). This link holds the starting address of the routine currently being used for single-character input—normally \$FD1B, the starting address of the standard input routine KeyIn.

When you issue an IN#n command from BASIC or an n CONTROL-K from the Monitor, the Apple IIc changes the link address in KSW to \$Cn00, the beginning of an I/O firmware subroutine. Subsequent calls for character input are thus transferred to that firmware. The firmware puts the input character, with its high bit set, into the accumulator and executes an RTS (return from subroutine) instruction to return control to the program that requested input.

When a disk operating system (DOS or ProDOS) is running, one or both of the standard I/O links hold addresses of the disk operating system's input and output routines. The operating system has internal locations that hold the addresses of the currently active character input and output routines.

▲Warning

If a program that is running with DOS or ProDOS changes the standard link addresses, either directly or via IN# and PR# commands, the operating system may be disconnected from the system. To avoid this problem, when programming in BASIC you should always issue an empty PRINT statement (to be sure that what follows begins a new line) before issuing the PRINT statement containing CONTROL-D and the IN# or PR# command.

The Monitor is discussed in Chapter 10.

Refer to the section on input and output link addresses in the operating system manuals for further details.

After changing either CSW or KSW, your assembly-language programs running under DOS should call the subroutine at location \$03EA. This subroutine transfers the link address to a location inside the operating system and then restores the operating system link address in the standard link location.

3.2 Standard Input Features

The Apple IIc's firmware includes two different subroutines for reading from the keyboard, RdKey (for read key) and GetLn (get line).

RdKey calls the current character input routine (that is, the one whose address is stored at KSW). This is normally KeyIn or C3KeyIn, which accepts one character from the keyboard. GetLn accepts a *sequence* of characters terminated with a carriage return. Thus GetLn allows line-oriented input using the current input routine.

GetLn also provides on-screen editing features: See Section 3.2.5.

3.2.1 RdKey Subroutine

A program can get a character from the keyboard by making a subroutine call to RdKey at memory location \$FD0C. RdKey passes control via the input link KSW to the current input subroutine, which is normally KeyIn.

RdKey displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COut routine, described below).

3.2.2 KeyIn Subroutine

KeyIn is the standard input subroutine. When your program calls it, KeyIn displays a cursor, waits until someone presses a key, then inserts the ASCII code of the key just pressed in the accumulator and returns to the calling program.

If the enhanced video firmware is inactive, KeyIn displays a cursor by alternately storing a checkerboard block in the cursor location, storing the original character, then storing the checkerboard again. If the firmware is active, C3KeyIn places a block cursor on the screen by inverting (swapping black for white) the character at the cursor position.

KeyIn also generates a random number. While it is waiting for the user to press a key, KeyIn repeatedly increments the 16-bit number in memory locations \$4E and \$4F. This number keeps increasing from 0 to \$FFFF (65535), then starts over again at 0. The value of this number changes so rapidly that it is very difficult to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a pseudo-random number routine.

3.2.3 GetLn Subroutine

Programs often need strings of characters as input. While you could call RdKey repeatedly to get several characters from the keyboard, there is an easier way to do it. The routine that you want to use in this case is named GetLn, and it starts at location \$FD6A. Using repeated calls to RdKey, GetLn accepts characters from the standard input subroutine—usually KeyIn—and puts them into the input buffer located in the memory page from \$0200 to \$02FF. GetLn also provides you with some basic on-screen editing and control features.

The first thing GetLn does when you call it is to display a prompt. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. Table 3-1 shows the prompt characters used by different programs on the Apple IIc.

GetLn uses the character stored at memory location \$33 as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect, because both BASIC interpreters and the Monitor restore it each time they request input from the user.

Section 3.2.5 describes other features of GetLn.

Table 3-1. Prompt Characters

Prompt Character	Program Requesting Input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 10)

Note: Applesoft uses GetLn1 (\$FD6F) when a program is executing. GetLn1 does not print a prompt.

As the user types each character, GetLn sends the character to the standard output routine—normally COut1—which displays it at the current cursor position and then advances the cursor to indicate the next character position. Control characters echoed by GetLn are not executed.

GetLn stores the characters in its buffer, starting at memory location \$0200 and using the X register to index the buffer. GetLn continues to accept and display characters until the user presses **RETURN** (or **CONTROL-X** to cancel the line). Then it clears the remainder of the line the cursor is on, stores the carriage-return code to mark the end of the buffer, places the cursor at the beginning of the next line, and returns.

The maximum line length that GetLn can handle is 255 characters. If the user types more than this, GetLn sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GetLn sounds a bell (tone) at every keypress after the 248th.

Note: The Applesoft interpreter accepts only 239 characters.

3.2.4 Escape Codes With GetLn

GetLn has many special functions that you invoke by typing escape codes on the keyboard. An escape code is sent by pressing **ESC**, releasing it, and then pressing some other key, as shown in Table 3-2.

Important! Be sure to release **ESC** right away. If you hold it too long, the auto-repeat mechanism begins, which may cancel the **ESC**.

Table 3-2. Escape Codes With GetLn

Escape Code	Function
ESC @	Clears the window and homes the cursor (places it in the upper-left corner of the screen), then exits from escape mode
ESC A or ESC a	Moves the cursor right one line and exits from escape mode
ESC B or ESC b	Moves the cursor left one line and exits from escape mode

Table 3-2—continued. Escape Codes With GetLn

Escape Code	Function
<code>ESC C</code> or <code>ESC c</code>	Moves the cursor down one line and exits from escape mode
<code>ESC D</code> or <code>ESC d</code>	Moves the cursor up one line; exits from escape mode
<code>ESC E</code> or <code>ESC e</code>	Clears to the end of the line; exits from escape mode
<code>ESC F</code> or <code>ESC f</code>	Clears to the bottom of the window; exits from escape mode
<code>ESC I</code> or <code>ESC i</code> or <code>ESC ↑</code>	Moves the cursor up one line; remains in escape mode*
<code>ESC J</code> or <code>ESC j</code> or <code>ESC ←</code>	Moves the cursor left one space; remains in escape mode*
<code>ESC K</code> or <code>ESC k</code> or <code>ESC →</code>	Moves the cursor right one space; remains in escape mode*
<code>ESC M</code> or <code>ESC m</code> or <code>ESC ↓</code>	Moves the cursor down one line; remains in escape mode*
<code>ESC 4</code>	Switches to 40-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†
<code>ESC 8</code>	Switches to 80-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†
<code>ESC CONTROL-D</code>	Disables control characters; only carriage return, line feed, bell, and backspace have an effect when printed
<code>ESC CONTROL-E</code>	Reactivates control characters
<code>ESC CONTROL-Q</code>	Deactivates the enhanced video firmware; sets links to KeyIn and COut1; restores normal window size (Table 3-5); exits from escape mode†

* Cursor-control key: see text.

† This code functions only when the enhanced video firmware is active.

In escape mode, you can keep using the arrow keys and the cursor movement keys **[I]**, **[J]**, **[K]**, and **[M]** without pressing **[ESC]** again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GetLn is in escape mode, it displays an inverse plus sign as the cursor. You leave escape mode by typing any key other than a cursor movement key.

Note: The escape codes with the arrow keys are the standard cursor movement keys on the Apple IIc. The escape codes with **[I]**, **[J]**, **[K]**, and **[M]** are the standard cursor movement keys on the Apple II and II Plus, and are present on the Apple IIc for compatibility.

Escape sequences can be used in the middle of an input line to change the appearance of the screen. They have no effect on the input line.

3.2.5 Editing With GetLn

For an introduction to editing with these features, refer to the *Applesoft Tutorial*.

Subroutine GetLn provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. Any program that uses GetLn for reading the keyboard has these features.

Cancel Line

Anytime you are typing a line, pressing **[CONTROL]-[X]** causes GetLn to cancel the line. GetLn displays a backslash (\) and issues a carriage return, then displays the prompt and waits for you to type a new line. GetLn takes the same action when you type more than 255 characters, as described above.




Backspace

When you press **[←]** (or **[CONTROL]-[H]**), GetLn moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COut, which moves the cursor back one space. If you type another character now, it replaces the character you backspaced over, both on the display and in the line buffer.

Each time you press **[←]**, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press **[←]** one more time, you have effectively canceled the line, and GetLn issues a carriage return and displays the prompt. The cursor moves even if the deleted character is an invisible control character. Thus it is possible for screen alignment and buffer alignment to be different.

See Section 3.2.4.

Retype

 (or **CONTROL**-) has a function that is complementary to the backspace function. When you press , GetLn picks up the character under the cursor just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display.

3.3 Standard Output Features

The standard output routine is named COut (character output). COut normally calls COut1 or C3COut1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COut1 and C3COut1 restrict their use of the display to an active area called the text window, described in Section 3.3.5.

3.3.1 COut Subroutine

Your program makes a subroutine call to COut at memory location \$FDED with a character in the accumulator. COut then passes control via the output link CSW to the current output subroutine, normally COut1 or C3COut1, which takes the character in the accumulator and writes it out. If the accumulator contains an uppercase or lowercase letter, a number, or a special character, COut1 or C3COut1 displays it; if the accumulator contains a control character, COut1 or C3COut1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COut1 or C3COut1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right-hand edge of the window, COut1 or C3COut1 moves it to the leftmost position on the next line down. If this would move the cursor position past the end of the last line in the window, COut1 or C3COut1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations \$24 and \$25. These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COut1 and C3COut1 do not display a cursor, but the input routines described above do, and they use this cursor position. However, changing CV directly does not change the cursor's vertical position until the next carriage return or reaching the end of the current line causes a call to VTab (for setting the base address within windows). If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COut1 or C3COut1.

▲Warning

When the video firmware is set for 80-column display, the value of CH is kept at 0 and the true horizontal position is stored at \$057B. When the 80-column video firmware is active, use \$057B instead of CH.

3.3.2 Control Characters With COut1

COut1 does not display control characters. Instead, the control characters listed in Table 3-3 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the appropriate escape code. The stop-list function, described separately, can only be invoked from the keyboard.

Escape codes are described in Section 3.2.4.

Table 3-3. Control Characters With COut1

Control Character	ASCII Name	Apple IIc Name	Action Taken by COut1
CONTROL-G	BEL	bell	Produces a 1000-Hz tone for 0.1 second
CONTROL-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
CONTROL-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed
CONTROL-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed

3.3.3 Control Characters With C3COut1

When the 80-column firmware is active, COut calls C3COut1 instead of COut1 for character output. C3COut1 does not display control characters, but you can use some control characters to control some of what the routine does. All other control characters are ignored.

The control characters listed in Table 3-4 are used to initiate some action by the firmware. Except for the stop-list function (**CONTROL-S**) you can send control characters to C3COut1 either from a program or from the Apple IIc's keyboard. The stop-list function can only be invoked from the keyboard. Most of the functions listed here can also be performed by using an equivalent escape code.

Escape codes are described in Section 3.2.4.

Table 3-4. Control Characters With C3COut1

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COut1
CONTROL-G	BEL	bell	Produces a 1000-Hz tone for 0.1 second
CONTROL-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
CONTROL-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed
CONTROL-K	VT	clear EOS	Clears from cursor position to the end of the screen*
CONTROL-L	FF	home and clear	Moves cursor position to upper-left corner of window and clears window*
CONTROL-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed
CONTROL-N	SO	normal	Sets display format normal*
CONTROL-O	SI	inverse	Sets display format inverse*
CONTROL-Q	DC1	40-column	Sets display to 40-column*
CONTROL-R	DC2	80-column	Sets display to 80-column*
CONTROL-S	DC3	stop-list	Stops listing characters on the display until another key is pressed†

Table 3-4—continued. Control Characters With C3COut1

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COut1
CONTROL-U	NAK	quit	Turns off enhanced video firmware*
CONTROL-V	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position*
CONTROL-W	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position*
CONTROL-X	CAN	disable MouseText	Disables MouseText character display; uses inverse uppercase
CONTROL-Y	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear)*
CONTROL-Z	SUB	clear line	Clears the line the cursor position is on*
CONTROL-[ESC	enable MouseText	Maps inverse uppercase characters to MouseText characters
CONTROL-\	FS	fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below*
CONTROL-]	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window)*
CONTROL-__	US	up	Moves cursor up a line, no scroll

* Doesn't work from the keyboard.

† Only works from the keyboard.

3.3.4 The Stop-List Feature

You can stop the Apple IIc from updating its display (if it is using either COut1 or C3COut1) by pressing **CONTROL-S**. Whenever COut1 or C3COut1 gets a carriage return from the program, it checks the keyboard for a **CONTROL-S**. If a **CONTROL-S** has been pressed, COut1 or C3COut1

stops and waits for another key to be pressed before resuming. The character code of the key that is pressed is ignored unless it is `CONTROL-C`, which is passed to the program. This feature lets you exit BASIC programs from stop-list mode.

3.3.5 The Text Window

The active portion of the display is called the text window. After you start up the computer or perform a reset, the entire display is the text window. COut1 or C3COut1 puts characters only into the window; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can restrict video activity to any rectangular portion of the display by changing the current text window. Your programs can thus control the placement of text in the display and protect other portions of the screen from being written over by new text. To do this, store the appropriate values into four locations in memory to set the top, bottom, left margin, and width of the text window. The following memory locations control the text window:

- The left margin is stored in memory location \$20. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal \$27); in an 80-column display, the maximum value is 79 (hexadecimal \$4F).
- The width of the text window is stored in memory location \$21. For a 40-column display, this value is normally 40 (hexadecimal \$28); for an 80-column display, it is normally 80 (hexadecimal \$50).
- The position of the top line of the text window is stored in memory location \$22. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal \$17).
- The position of the bottom line of the screen plus 1 is stored in memory location \$23. It is normally 24 (hexadecimal \$18) for the bottom line of the display. Its minimum value is 1.

Important! | Pascal does not use this method of supporting window widths.

▲Warning | Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80 columns). If this happens, COut1 or C3COut1 may put characters into memory locations outside the display page, possibly destroying programs or data.

Table 3-5 summarizes the memory locations and the possible values for the text window parameters.

Table 3-5. Text Window Memory Locations

Window Parameter	Location		Minimum Value		Normal Values				Maximum Values			
	Dec	Hex	Dec	Hex	40-col.		80-col.		40-col.		80-col.	
Left Edge	32	\$20	00	\$00	00	\$00	00	\$00	39	\$27	79	\$4F
Width	33	\$21	00	\$00	40	\$28	80	\$50	40	\$28	80	\$50
Top Edge	34	\$22	00	\$00	00	\$00	00	\$00	23	\$17	23	\$17
Bottom Edge	35	\$23	01	\$01	24	\$18	24	\$18	24	\$18	24	\$18

3.3.6 Normal, Inverse, and Flashing Text

The way that the Apple IIc displays characters is affected by two things: the value that is stored in the inverse flag (zero page location \$32), and whether the enhanced video firmware is off or on. The inverse flag's influence is discussed in the next two subsections.

If the enhanced video firmware is off, the Apple IIc displays what is called the primary character set; if the video firmware is on, the Apple IIc displays what is called the alternate character set.

The primary character set includes normal (light on dark), inverse (dark on light), and flashing (alternating normal and inverse) characters. Lowercase inverse characters are not included in the primary character set.

The alternate character set includes normal and inverse characters (including lowercase inverse), and a set of graphic characters called MouseText. Flashing characters are not included in the alternate character set.

If you want your program to display a character, it should first load the character to be displayed in the accumulator, and then call the character-output subroutine COut. For example, to display the character corresponding to \$C8, you can use something like this:

```
LDA #$C8
JSR COut
```

Both these display character sets are described in Chapter 5.

Primary Character Set Display

The primary character set is displayed by COut1, which operates only when the enhanced video firmware is off. The primary character set includes text in normal, inverse, or flashing format, but not inverse or flashing *lowercase* text.

For a brief explanation of logical functions, refer to Appendix H.

If the value of the character sent to COut1 is greater than or equal to \$A0, that value is logically ANDed with the value of the inverse flag (at location \$32), then displayed. (If you're curious about which ASCII character is being sent, subtract \$80 from the value being sent to COut1.) You can use the following inverse flag values:

- ☐ \$FF (decimal 255) produces the normal character format.
- ☐ \$3F (decimal 63) produces the inverse character format.
- ☐ \$7F (decimal 127) produces the flashing character format.

Important!

To avoid unusual character display results, use only the three values \$3F, \$7F, and \$FF.

COut1 interprets character values from \$80 through \$9F as control characters and tries to execute them.

Character values from \$00 through \$7F are all interpreted as display characters, not control characters.

Alternate Character Set Display

The alternate character set includes normal and inverse format characters and the MouseText graphic characters. You should use C3COut1, the standard output link when the enhanced video firmware is *active*, to display the alternate character set. Here are the rules for using the alternate character set:

- ☐ Control characters are not displayed. Characters sent to C3COut1 are interpreted as control characters if they are in the range \$00 through \$1F or \$80 through \$9F.
- ☐ Characters in the range \$20 through \$7F and \$A0 through \$FF are displayed.
- ☐ If inverse flag (location \$32) bit 7 is 1, the character is normal.
- ☐ If inverse flag bit 7 is 0, the character is inverse.
- ☐ If MouseText is off, characters \$40 through \$5F are remapped to the range \$00 through \$1F and are displayed as uppercase inverse characters.
- ☐ If MouseText is on, character values \$40 through \$5F are left unchanged, and the characters are displayed as MouseText.

MouseText is described more fully in Chapter 5.

See Section 5.2.2.

3.4 Port I/O

The Apple IIc is a member of the Apple II family of computers; however, unlike the Apple II, II Plus, and IIe, the Apple IIc does not have peripheral connector slots. In place of these, it has ports—the equivalent of firmware interface cards installed in slots.

3.4.1 Standard Link Entry Points

To maintain compatibility with existing software and its protocols, each port's I/O firmware has the same standard entry points (\$Cn00) as its equivalent slot in another Apple II would have. Table 3-6 shows these equivalents, as well as listing the chapter where each port is described.

Section 3.1 describes how and when these entry addresses are placed in CSW and KSW. For example, issuing PR#n or IN#n changes the output and input links, respectively, so that subsequent output or input is handled by the firmware starting at address \$Cn00, and thus goes to or comes from the selected device.

Table 3-6. Port Characteristics

Port	Entry Point	Port Connector	Use	Chapter
1	\$C100	Serial port 1	Printers	7
2	\$C200	Serial port 2	Communication	8
3	\$C300	Video connectors	Enhanced video firmware	5
4	\$C400	Mouse	Mouse	9
5	\$C500	Intelligent disk port devices		
6	\$C600	Disk drives	Built-in and external drives	6
7	\$C700	No device	Reserved	6

Important!

The addresses shown in Table 3-6 are not entry points in the sense that you can send characters to be printed by sending them to JSR \$Cn00.

3.4.2 Firmware Protocol

The Apple IIc supports a standard firmware protocol that, in addition to the standard link address, provides a table of device identification and entry points to standard and optional firmware subroutines. The protocol is equivalent to the Pascal 1.1 firmware protocol in use on other Apple II's, and is outlined in Table 3-7.

Table 3-7. Firmware Protocol Locations

Address	Value	Description
\$Cn05	\$38	Pascal firmware card/port identifier
\$Cn07	\$18	Pascal firmware card/port identifier
\$Cn0B	\$01	Generic signature byte of a firmware card/port
\$Cn0C	\$ci	Device signature byte: i is an identifier (not necessarily unique). c = device class (not all used on the Apple IIc): \$00 reserved \$01 printer \$02 hand control or other X-Y device \$03 serial or parallel I/O card/port \$04 modem \$05 sound or speech device \$06 clock \$07 mass-storage device \$08 80-column card/port \$09 network or bus interface \$0A special purpose (none of the above) \$0B-0F reserved
\$Cn0D	ii	\$Cnii is the initialization entry address (PInit).
\$Cn0E	rr	\$Cnrr is the read routine entry address (PRead) (returns character read in A register).
\$Cn0F	ww	\$Cnww is the write routine entry address (PWrite) (enters with character to write in A register).

Table 3-7—continued. Firmware Protocol Locations

Address	Value	Description
\$Cn10	ss	\$Cnss is the status routine entry address (PStatus) (enters with request code in A register: 0 to ask “Are you ready to accept output?” or 1 to ask “Do you have input ready?”).
\$Cn11	\$00	if additional address bytes follow; nonzero if not

Each table begins with identification bytes (\$Cn05 through \$Cn0C). Then, starting with address \$Cn0D, each byte in the table represents the low-order byte of the entry-point address of a firmware routine. The high-order byte of each address is always \$Cn, where n is the port number. Your program uses these byte values to construct its own jump table for subroutine calls to the ports.

All port routines require, on entry, that the X register contain \$Cn and that the Y register contain \$n0.

All routines, on exit, return an error code in the X register (0 means no error occurred; 3 means the request was invalid). The carry bit in the program status register usually contains a reply to a request code (0 means no; 1 means yes).

All the Apple IIc ports except the disk port conform to this protocol. The disk port is described in Chapter 6.

3.4.3 Port I/O Space

By a convention used in other Apple II series machines, each port or slot has exclusive use of 16 memory locations set aside for data input and output. The addresses of these locations are of the form \$C080 + #n0, where n is the port or slot number. Table 3-8 lists the port I/O space used in the Apple IIc.

For more information, refer to the hardware page memory map in Appendix B.

Table 3-8. Port I/O Locations

Port	Locations
1	\$C090–\$C09F
2	\$C0A0–\$C0AF
6	\$C0E0–\$C0EF

3.4.4 Port ROM Space

In the Apple II and IIe, one 256-byte page of memory space is allocated to each slot. This space is used for read-only memory (ROM or PROM on the interface card) with driver programs that control the operation of input/output devices, as outlined in Table 3-7. On the Apple IIc, this space is dedicated to port firmware. However, I/O ROM space in the Apple IIc is used as efficiently as possible, and there is not a strict correspondence between firmware for port *n* and the \$C*n*00 space, except as regards entry points.

3.4.5 Expansion ROM Space

The 2K-byte memory space from \$C800 to \$CFFF in the Apple IIc—called expansion ROM space on the Apple II, II Plus, and IIe—contains the enhanced video firmware and port and memory transfer subroutines. The Apple IIc, unlike the II, II Plus, or IIe, always has this space switched in.

3.4.6 Port Screen Hole RAM Space

There are 128 bytes of memory (64 in main memory, 64 in auxiliary memory) allocated to the ports, 8 bytes per port, as shown in Table 3-9. These bytes are reserved for use by the system, except as described in Chapters 4 through 9.

Table 3-9. Port Screen Hole Memory Locations

Base Address	Ports:						
	1	2	3	4	5	6	7
\$0478	\$0479	\$047A	\$047B	\$047C	\$047D	\$047E	\$047F
\$04F8	\$04F9	\$04FA	\$04FB	\$04FC	\$04FD	\$04FE	\$04FF
\$0578	\$0579	\$057A	\$057B	\$057C	\$057D	\$057E	\$057F
\$05F8	\$05F9	\$05FA	\$05FB	\$05FC	\$05FD	\$05FE	\$05FF
\$0678	\$0679	\$067A	\$067B	\$067C	\$067D	\$067E	\$067F
\$06F8	\$06F9	\$06FA	\$06FB	\$06FC	\$06FD	\$06FE	\$06FF
\$0778	\$0779	\$077A	\$077B	\$077C	\$077D	\$077E	\$077F
\$07F8	\$07F9	\$07FA	\$07FB	\$07FC	\$07FD	\$07FE	\$07FF

These addresses are unused bytes in the RAM reserved for text and low-resolution graphics displays, and hence they are sometimes called screen holes. These particular locations are not displayed on the screen and their contents are not changed by the built-in output routines. In other words, they are used by the output routines but they are not part of the video display.

▲ **Warning**

All the screen holes in auxiliary memory, and many of them in main memory, are reserved for special use by Apple IIc firmware—for example, to store initialization information. Do not use any locations marked **reserved** in this manual.

The way that port firmware uses these RAM locations and their addresses is covered in Chapters 4 through 10.

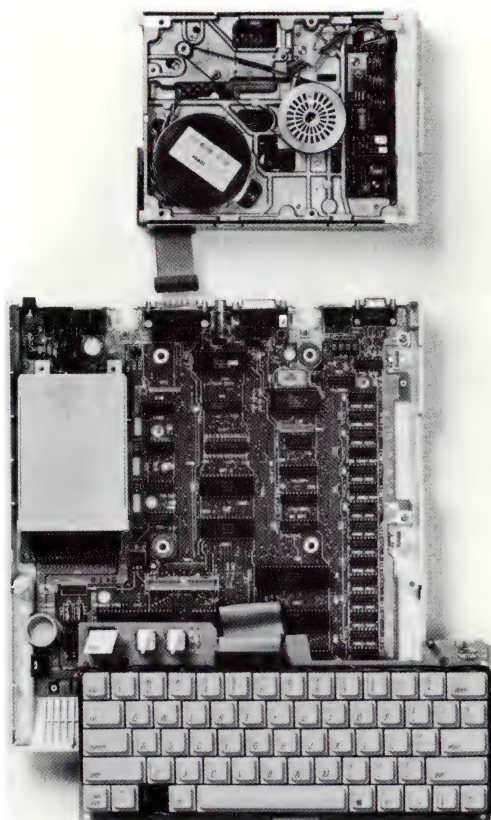
3.5 Interrupts

Appendix E describes interrupt handling on the Apple IIc.

Interrupts are a way to more efficiently use the hardware in a computer. Interrupt support built into the Apple IIc's firmware is described briefly below.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers program control through the vector in locations \$FFFE through \$FFFF of ROM or whichever bank of RAM is switched in (Chapter 2). If ROM is switched in, this vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an interrupt that should be handled internally. If so, the Monitor handles it and then returns control to the interrupted program.

If the interrupt is due to a BRK (\$00) instruction, control is transferred through the BRK vector (\$03F0–\$03F1). Otherwise, control is transferred through the IRQ vector (\$03FE–\$03FF).



This chapter describes how to use two of the Apple IIc's built-in devices: the keyboard and the speaker.

4.1 Keyboard Input

Table 4-1 describes the characteristics of the keyboard that relate to programming. You won't have to write routines to read the keyboard from all your assembly-language programs since the Apple IIc firmware Monitor provides keyboard support through the three standard input routines described in Chapter 3—RdKey, KeyIn, and GetLn. You *can* do all your keyboard handling directly in your programs if you want to, but it's nice to know that you're not forced to.

4.1.1 Reading the Keyboard

The keyboard encoder and ROM (see Chapter 11) can generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Your machine-language programs can call RdKey to get characters from the keyboard. RdKey reads characters a byte at a time from the keyboard data location (\$C000) shown in Table 4-1.

Here is how your programs should go about reading the keyboard:

1. Test bit 7 of address \$C000 to see if a key has been pressed. Bit 7 is the keyboard strobe bit.
2. When bit 7 goes to a 1, you know that the low-order seven bits of \$C000 are a valid character.
3. Clear the keyboard strobe (bit 7) at \$C000 by reading or writing *anything* to address \$C010.

\$C010 has another function besides clearing the keyboard strobe: its high bit is a 1 while a key is pressed (except the Apple keys, **CONTROL**, **SHIFT**, **CAPS LOCK**, and **RESET**). Bit 7 at this location is therefore called any-key-down. You could use this to let a program do something useful other than just waiting for the next key to be pressed. (People are generally a *lot* slower than the Apple IIc.) Check \$C010 occasionally to see **if** something should be done.

Important!



If your program needs to read both the keyboard flag and strobe, it **must** read the strobe bit first. Anytime you read the any-key-down bit at \$C010, you also clear the keyboard strobe bit at \$C000.

For a description of how the keyboard strobe works, refer to Appendix E.

Table 4-1. Keyboard Input Characteristics

Port number:	None
Commands:	Keyboard is always on, in the sense that any keypress generates a KSTRB.
Initial characteristics:	Reset routine clears the keyboard strobe and sets the keyboard as the standard input device (that is, sets KSW to point to RdKey).

Hardware locations:

	\$C000	Keyboard data and strobe
	\$C010	Any-key-down flag and clear-strobe switch
	\$C060	40-column switch status on bit 7; 1 = 40-column display = switch down
Game input switches: See Chapter 9.	\$C061	 status on bit 7; 1 = pressed (also game input switch 0)
	\$C062	 status on bit 7; 1 = pressed

Monitor firmware routines:

	Location	Name	Description
GetLn, GetLn1, and RdKey: See Chapter 3.	\$FD6A	GetLn	Gets an input line with prompt
	\$FD67	GetLnZ	Gets an input line with preceding carriage return
	\$FD6F	GetLn1	Gets an input line, but with no preceding prompt
	\$FD1B	KeyIn	The keyboard input subroutine
	\$FD35	RdChar	Gets an input character or escape code
	\$FD0C	RdKey	The standard character input subroutine

Use of other pages:

Page 2	The standard character string input buffer (see GetLn description)
--------	--

After your program has cleared the keyboard strobe, the strobe remains low until another key is pressed.

Table 4-2 shows the ASCII codes generated by all the keys on the Apple IIc keyboard. Remember, if the strobe bit is set, the character values that your program sees will be equal to the values given in Table 4-2 plus \$80.

Table 4-2. Keys and ASCII Codes
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.










Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
 DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
 ←	08	BS	08	BS	08	BS	08	BS
 TAB	09	HT	09	HT	09	HT	09	HT
 ↓	0A	LF	0A	LF	0A	LF	0A	LF
 ↑	0B	VT	0B	VT	0B	VT	0B	VT
 RETURN	0D	CR	0D	CR	0D	CR	0D	CR
 →	15	NAK	15	NAK	15	NAK	15	NAK
 ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
 SPACE	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C		2C	,	3C	<	3C	<
- _	2D		1F	US	5F	_	1F	US
. >	2E		2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29)	29)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (39	9	39	9	28	(28	(
	3B		3B		3A		3A	

Table 4-2—continued. Keys and ASCII Codes*Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.*





Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] }	5D]	1D	GS	7D	}	1D	GS
` ~	60	`	60	`	7E	~	7E	~
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB


Keystrokes can also generate interrupts: See Appendix E.

This restarting process is called the **reset routine**, and is described in Chapter 2.

For information on how to have programs interpret keystrokes in a standard way, refer to the *Apple II Design Guidelines* listed in the Bibliography.

There are several keys that do not generate ASCII codes themselves, but alter the characters produced by other keys. These modifier keys are **CONTROL**, **SHIFT**, and **CAPS LOCK**.

Your programs can also use the  and  as character modifier keys while handling keyboard input, and, if one or both of them is pressed, branch to a special routine, such as a help program. Your program can read  at \$C061 and  at \$C062.

Another key that doesn't generate a code is **RESET**, located at the upper-left corner of the keyboard; it is connected directly to the Apple IIc's processor. Pressing **RESET** with **CONTROL** depressed normally causes the system to stop whatever program it's running and restart itself. If you hold  while pressing **CONTROL-RESET**, the Apple IIc performs a forced cold start. The restart sequence is described in Chapter 2.

4.1.2 Monitor Firmware Support for Keyboard Input

Chapter 3 describes the three standard Monitor input routines serving the keyboard: GetLn, RdKey, and KeyIn. This section discusses the three other available Monitor routines.

GetLnZ


GetLnZ (at address \$FD67) is an alternate entry point for GetLn that first sends a carriage return to the standard output, then continues into GetLn.

GetLn1

GetLn1 (at address \$FD6F) is an alternate entry point for GetLn that does not issue a prompt before it accepts the input line. However, if the user cancels the input line with too many backspaces or with **CONTROL-X**, then GetLn1 issues the prompt stored at location \$33 when it gets another line.

RdChar

RdChar (at address \$FD35) is a subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.

If the enhanced video firmware is active,  (**CONTROL-U**) reads a character from the screen as if it were typed from the keyboard. This is a function of the Monitor's built-in editing capability described in Chapter 3.

Electrical specifications of the speaker circuit appear in Chapter 11.

4.2 Speaker Output

The Apple IIc has a small speaker mounted near the front of the bottom plate of its case. The speaker is connected to a soft switch that toggles; that is, the switch has two states, off and on, and it changes from one to the other each time it is accessed. Table 4-3 describes the speaker output characteristics.

Table 4-3. Speaker Output Characteristics

Port number:	None	
Commands:	Some programs sound the speaker in response to CONTROL-G.	
Initial characteristics:	Reset routine sounds the speaker.	
Hardware location:		
\$C030	Toggle speaker (read only)	
Monitor firmware routines:		
Location	Name	Description
\$FBDD	Bell1	Sends a beep to the speaker
\$FF3A	Bell	Sends CONTROL-G to the current output

4.2.1 Using the Speaker

If you switch the speaker once, by reading or writing to \$C030, it emits a click; to make longer sounds, access the speaker repeatedly. The switch for the speaker uses memory location \$C030. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

Important!

You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.

See Chapter 3.

4.2.2 Monitor Firmware Support for Speaker Output

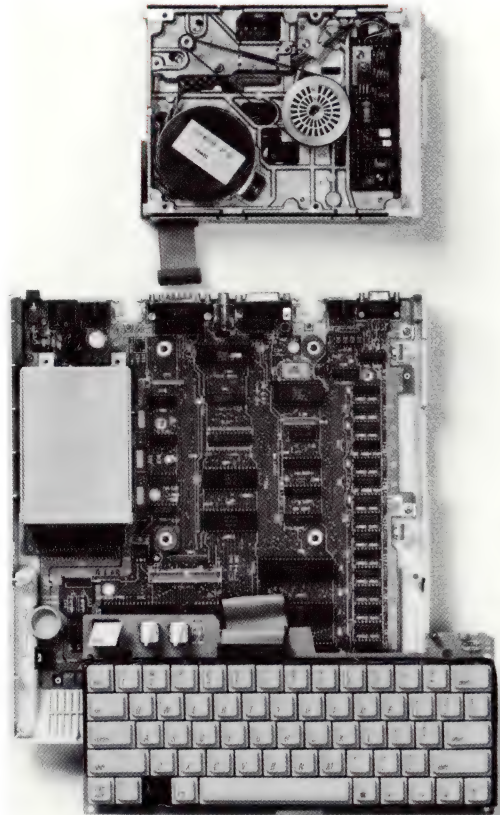
The Monitor supports the speaker with one simple routine, Bell1. A related routine, Bell, supports the current output device—the one that CSW points to.

Bell1

Bell1 (at address \$FDDB) makes a beep through the speaker by generating a 1-kHz tone in the Apple IIc's speaker for 0.1 second. This routine scrambles the A and X registers.

Bell

The Monitor routine Bell (at location \$FF3A) writes a bell control character (ASCII CONTROL-G) to the current output device. This routine leaves the accumulator holding \$87.



NTSC stands for National Television Standards Committee, a group that formulates broadcast and reception guidelines used by the USA and several other countries.

The Apple IIc's primary output device is its video display. You can use any ordinary color or monochrome video monitor with the Apple IIc. An ordinary monitor is one that accepts NTSC-compatible composite video. If you use Apple IIc color graphics with a black-and-white monitor, the display will appear as black, white, and two shades of gray.

If you are only using graphics modes and 40-column text, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your Apple IIc; otherwise, you must attach an RF video modulator between the Apple IIc and the television set.

Important!

The Apple IIc can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

Table 5-1 summarizes the video output port's characteristics and points to other information in this chapter.

Table 5-1. Video Output Port Characteristics

Port number:	Output port 3
Commands:	See Figure 5-3
Initial characteristics:	See Figure 5-3 Note: If a program is to use the enhanced video firmware, it should turn it on and then immediately check the 80/40 switch. If the switch is in the 40 position, the program should issue a CONTROL-Q.
Hardware locations:	See Table 5-7
Monitor firmware routines:	See Table 5-11
I/O firmware entry points:	See Table 5-12

5.1 Video Display Specifications

Table 5-2 summarizes the video display’s specifications, and provides a further guide to other information in this chapter.

Table 5-2. Video Display Specifications

Display modes:	40-column text; map: Figure 5-5
	80-column text; map: Figure 5-6
	Low-resolution color graphics; map: Figure 5-7
	High-resolution color graphics; map: Figure 5-8
	Double-high-resolution color graphics; map: Figure 5-9
Text capacity:	24 lines by 80 columns (character positions)
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, inverse, flashing, MouseText (Table 5-3)
Low-resolution graphics:	16 colors (Table 5-4): 40 horizontal by 48 vertical; map: Figure 5-7
High-resolution graphics:	6 colors (Table 5-5): 140 horizontal by 192 vertical (restricted)
	Black-and-white: 280 horizontal by 192 vertical; map: Figure 5-8
Double-high-resolution graphics:	16 colors (Table 5-6): 140 horizontal by 192 vertical (no restrictions)
	Black-and-white: 560 horizontal by 192 vertical; map: Figure 5-9

The video signal produced by the Apple IIc is NTSC-compatible composite color video available at two places on the back panel of the Apple IIc: the RCA-type phono jack and the 15-pin D-type connector. Use the RCA-type phono jack to connect a video monitor, and the DB-15 connector for an external video modulator or other video expansion hardware.

See Section 11.9.5 for more on video expansion hardware.

5.2 Text Modes

Either of the Apple IIc's two text modes can display all 96 ASCII characters: uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide (with a few exceptions, such as underscore), leaving two blank columns of dots between characters in a row. Except for lowercase letters with descenders, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other monochrome color used by your monitor) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called inverse format.

5.2.1 Text Character Sets

The Apple IIc can display either of two text character sets: the primary set and an alternate set (Table 5-3). The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- ☐ normal, with white dots on a black screen
- ☐ inverse, with black dots on a white screen
- ☐ flashing, alternating between normal and inverse.

The Apple IIc can display uppercase characters in all three formats—normal, inverse, and flashing—with the primary character set. Lowercase letters can only be displayed in normal format. This makes the primary character set compatible with most software written for the Apple II and II Plus, which can display text in flashing format but don't have lowercase characters.

The alternate character set trades the flashing format for a complete set of inverse characters. With the alternate character set, the Apple IIc can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display MouseText.

MouseText: See Section 5.2.2.

You can select between character sets with the alternate-text soft switch, described in Section 5.6. Table 5-3 shows the character codes in decimal and hexadecimal for the Apple IIc primary and alternate character sets in normal, inverse, and flashing formats.

To identify particular characters and values, refer to Table 4-2.

Table 5-3. The Display Character Sets

Hex Values	Primary Character Set		Alternate Character Set	
	Character Type	Format	Character Type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText	
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special characters	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select format and the group within ASCII (Section 3.3.6).

5.2.2 MouseText

The alternate character set contains 32 graphics characters called MouseText in place of the primary set's inverse uppercase characters from \$40 through \$5F. These graphics are especially convenient to use with a mouse, since they can be generated by character codes instead of groups of high-resolution byte values, and they can be moved around quickly. To use MouseText characters, do the following:

1. Turn on the enhanced video firmware with PR#3 or 6 CONTROL-P.
2. Set inverse mode: use the INVERSE command or put \$3F in location \$32, or print CONTROL-O.
3. Turn on MouseText with PRINT CHR\$(27); or pass \$1B to COut in the accumulator.
4. Print the uppercase letter (or other ASCII character in the range \$40 through \$5F: @ [\] ^ or _) that corresponds to the MouseText character you want.
5. Turn off MouseText with PRINT CHR\$(24); or pass \$18 to COut1 in the accumulator.

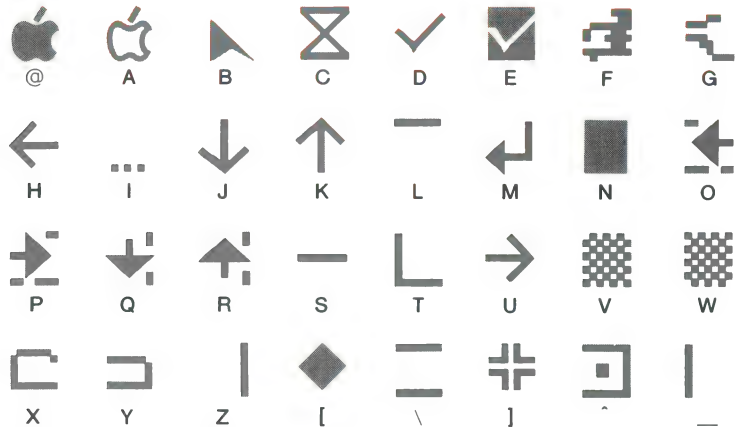
6. Set normal mode: use the NORMAL command or put \$FF in location \$32, or print a CONTROL-N.

Here is a sample Applesoft program that prints all the MouseText characters:

```
10 D$=CHR$(4)
20 PRINT PRINT D$;"PR#3"
30 INVERSE
40 PRINT CHR$(27);"@ABCDEFGHIJKLMNPOQRSTUVWXYZ[]^_";
50 PRINT CHR$(24);
60 NORMAL
```

MouseText characters and their corresponding ASCII characters are shown in Figure 5-1.

Figure 5-1. MouseText Characters



5.2.3 40-Column Versus 80-Column Text

The Apple IIc has two text display modes: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 5-2. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

Figure 5-2. 40-Column and 80-Column Text (With Alternate Character Set)

```
LIST 0,100
```

```
10 REM APPLESOFT CHARACTER DEMO
20 TEXT : HOME
30 PRINT : PRINT "Applesoft Char
  acter Demo"
40 PRINT : PRINT "Which character
  set—"
50 PRINT : INPUT 'Primary (P) or
  Alternate (A) ?' A$
60 IF LEN (A$) < 1 THEN 50
65 LET A$ = LEFT$ (A$,1)
70 IF A$ = "P" THEN POKE 49166,
  0
80 IF A$ = "A" THEN POKE 49167,
  0
90 PRINT : PRINT "...printing th
  e same line, first"
100 PRINT ' in NORMAL, then INVE
  RSE, then FLASH:": PRINT
```

```
]
```

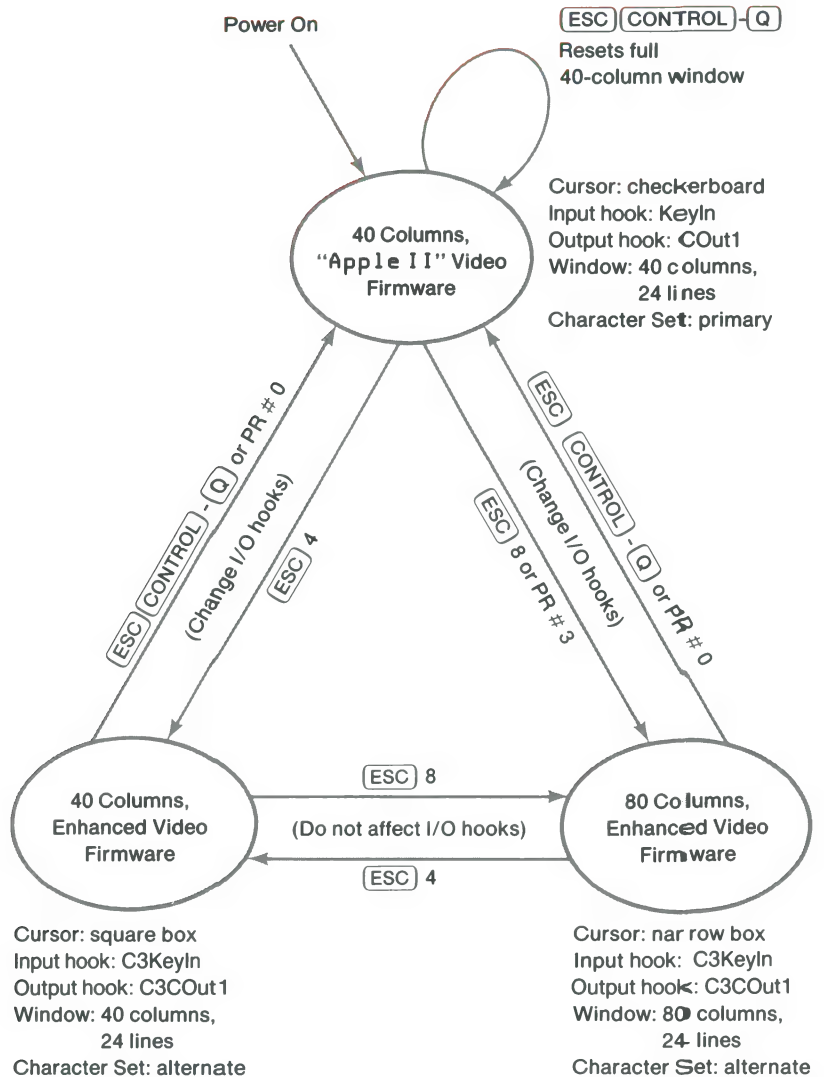
```
LIST 0,100
```

```
10 REM APPLESOFT CHARACTER DEMO
20 TEXT : HOME
30 PRINT ; PRINT "Applesoft Character Demo"
40 PRINT : PRINT "Which character set—"
50 PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60 IF LEN (A$) < 1 THEN 50
65 LET A$ = LEFT$ (A$,1)
70 IF A$ = "P" THEN POKE 49166,0
80 IF A$ = "A" THEN POKE 49167,0
90 PRINT : PRINT "...printing the same line, first"
100 PRINT " in NORMAL, then INVERSE, then FLASH:": PRINT
```

```
]
```


Figure 5-3 shows the characteristics of the text display modes and how to switch between them.

Figure 5-3. Text Mode Characteristics and Switching



5.3 Graphics Modes

The Apple IIc can produce color video graphics in any of three different modes:

- low-resolution graphics, 48 rows by 40 columns
- high-resolution graphics, 192 rows by 280 columns
- double-high-resolution graphics, 192 rows by 560 columns

Each graphics mode treats the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes on the screen; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

5.3.1 Low-Resolution Graphics

The Apple IIc displays an array of 48 rows by 40 columns of colored blocks in the low-resolution graphics mode. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and two shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

The low-resolution graphics display data are stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 5-4. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value \$D8 produces a brown block atop a yellow block on the screen.

Table 5-4. Low-Resolution Graphics Colors
Note: Colors may vary, depending on adjustment of monitor or television set.

Nibble Value			Nibble Value		
Decimal	Hex	Color	Decimal	Hex	Color
0	\$00	black	8	\$08	brown
1	\$01	magenta	9	\$09	orange
2	\$02	dark blue	10	\$0A	gray 2
3	\$03	purple	11	\$0B	pink
4	\$04	dark green	12	\$0C	light green
5	\$05	gray 1	13	\$0D	yellow
6	\$06	medium blue	14	\$0E	aquamarine
7	\$07	light blue	15	\$0F	white

As explained in Section 5.5, the text display and the low-resolution graphics display use the same area in memory. Your programs should usually clear this part of memory when they change display modes, but you can store data as text and display them as graphics, or vice versa. All you have to do is change the mode switch, described in Section 5.6, without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex low-resolution graphics displays quickly.

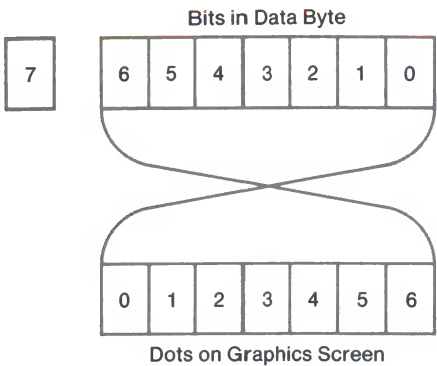
5.3.2 High-Resolution Graphics

In the high-resolution graphics mode, the Apple IIc displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described below, by the color of adjacent dots. Adjacent dots of the same color merge to form a continuous colored area.

High-resolution graphics display data are stored in either of two 8192-byte areas in memory. These areas are called high-resolution Page 1 and Page 2; think of them as display data buffers. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

The Apple IIc high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Apple IIc's memory. The seven low-order bits of each display byte control a row of seven adjacent dots on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) dots. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the next-least significant bit, and so on, as shown in Figure 5-4.

Figure 5-4. High-Resolution Display Bits



There is a simple correspondence between bits in memory and dots on the screen on a black-and-white monitor. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots merge together; alternating black and white dots merge to a continuous gray.

A dot whose controlling bit is off (0) is black on an NTSC color monitor or a color television set. If the bit is on, the dot is white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the leftmost column of dots column 0, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green—but only if the dots on either side are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange (again, only if the dots on either side are black). Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In brief, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- Dots in even-numbered columns can be black, purple, or blue.
- Dots in odd-numbered columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 5-5. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 5-5. High-Resolution Graphics Colors
Note: Colors may vary, depending on adjustment of monitor or television set.

Bits 0–6	Bit 7 Off	Bit 7 On
Adjacent columns off	black 1	black 2
Even columns on	purple	blue
Odd columns on	green	orange
Adjacent columns on	white 1	white 2

For more details about the way the Apple IIc produces color on a TV set, see Chapter 11. For a table of reversed bit patterns, refer to Appendix H.

The peculiar behavior of the high-resolution colors reflects in part the way NTSC color television works. The dots that make up the Apple IIc video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating on and off dots at this spacing cause a color monitor or TV set to produce color, but two or more on dots together do not.

5.3.3 Double-High-Resolution Graphics

The horizontal resolution of double-high-resolution graphics is 560 dots per line, with 192 lines. Double-high-resolution graphics maps the low-order seven bits of the bytes in the two double-high-resolution graphics pages. A double-high-resolution page is made up of a 8192-byte page in main memory and an equivalent page having the same address in auxiliary memory. In most cases, only the first double-high-resolution graphics page is used.

The bytes in the main-memory and auxiliary-memory pages are displayed in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. A dot whose controlling bit is off (0) is black when displayed.

Unlike high-resolution color (Section 5.3.2), double-high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a four-dot-wide window moving across the screen: at any given time, the color displayed corresponds to the 4-bit value from Table 5-6 that corresponds to the window's position (Figure 5-9). Effective horizontal resolution with color is 140 (560 divided by 4).

Table 5-6, on the next page, describes the data values used to produce colors in double-high-resolution graphics. To use the table, divide the column number by 4, and use the remainder to find the correct column: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on), *mb1* is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 2, 9 and so on), and similarly for *ab2* and *mb3*.

5.4 Mixed-Mode Displays

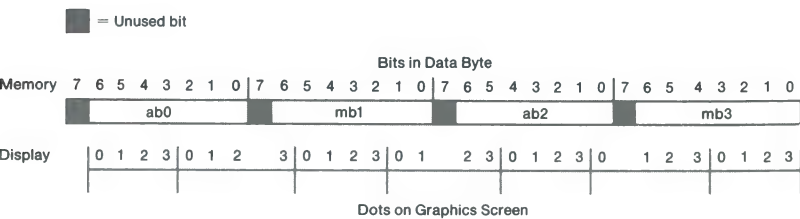
Any of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called mixed-mode displays. To use them, the TEXT switch must be off (read \$C050) and the MIXED switch on (read \$C053).

Important! | You cannot display 40-column text with double-high-resolution graphics.

To determine what appears where in mixed-mode displays, refer to Figures 5-5 through 5-9 in Section 5.7. See the bottom sixth of the appropriate text display (Figure 5-5 or 5-6) and the upper five-sixths (down to the heavy horizontal line) in the appropriate graphics display (Figures 5-7 to 5-9).

Table 5-6. Double-High-Resolution Graphics Colors
Note: Colors may vary, depending on adjustment of monitor or television set.

Color	ab0	mb1	ab2	mb3	Repeated Bit Pattern
black	\$00	\$00	\$00	\$00	0000
magenta	\$08	\$11	\$22	\$44	0001
brown	\$44	\$08	\$11	\$22	0010
orange	\$4C	\$19	\$33	\$66	0011
dark green	\$22	\$44	\$08	\$11	0100
gray 1	\$2A	\$55	\$2A	\$55	0101
green	\$66	\$4C	\$19	\$33	0110
yellow	\$6E	\$5D	\$3B	\$77	0111
dark blue	\$11	\$22	\$44	\$08	1000
purple	\$19	\$33	\$66	\$4C	1001
gray 2	\$55	\$2A	\$55	\$2A	1010
pink	\$5D	\$3B	\$77	\$6E	1011
medium blue	\$33	\$66	\$4C	\$19	1100
light blue	\$3B	\$77	\$6E	\$5D	1101
aqua	\$77	\$6E	\$5D	\$3B	1110
white	\$7F	\$7F	\$7F	\$7F	1111



5.5 Display Pages

The Apple IIc uses data stored in specific areas in memory to generate its video displays. These areas, called display pages, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object—a character, a colored block, or a group of adjacent dots—at a certain location on the display, depending on the current display mode.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called text Page 1 and text Page 2, and they are located at \$0400 through \$07FF and \$0800 through \$0BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch between displays. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four lines of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages when the enhanced video firmware is active. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory. This additional memory is *not* the same as text Page 2—in fact, it is text Page 1X, and it occupies the same address space as text Page 1 (see Figure 2-11). The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use these routines for all normal text output.

Important!

The built-in video firmware always displays Page 1 text. You cannot write text to Page 2 with the built-in firmware.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage as a low-resolution display, as shown in Table 5-7.

The double-high-resolution graphics mode interleaves the two high-resolution pages (Pages 1 and 1X) in exactly the same way as 80-column text mode interleaves the text pages: column 0 and all subsequent even-numbered columns come from the auxiliary page; column 1 and all subsequent odd-numbered columns come from the main page.

Table 5-7. Video Display Page Locations

Display Mode	Display Page	Lowest Address		Highest Address	
40-column text, low-resolution graphics	1	\$0400	1024	\$07FF	2047
	2*	\$0800	2048	\$0BFF	3071
80-column text	1	\$0400	1024	\$07FF	2047
	2*	\$0800	2048	\$0BFF	3071
High-resolution graphics	1	\$2000	8192	\$3FFF	16383
	2	\$4000	16384	\$5FFF	24575
Double-high- resolution graphics	1†	\$2000	8192	\$3FFF	16383
	2†	\$4000	16384	\$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to Section 5.6.

† See Section 5.3.3.

5.6 Display Mode Switching

Table 5-8 shows the reserved locations for the soft switches that control the different display modes. The column of the table labeled *Action* indicates what to do to activate or read a switch setting: R means read the location, W means write anything to the location, R/W means read or write, and R7 means read the location and then check bit 7.

Table 5-9 lists the display modes that the firmware can set up automatically. In the 40-column modes, the contents of the standard I/O hooks KSW and CSW (Chapter 3) determine whether the enhanced video firmware features are available or not. The firmware also takes care of setting or clearing AltChar.

Table 5-10 lists other display modes available but not supported by firmware. For modes that display Page 2 with the 80Col switch on, your program may have to turn 80Store off after the firmware has turned it on.

Double-low-resolution shows on the display screen when HiRes is off and both 80Col and DHiRes are on. It is the low-resolution graphics equivalent of 80-column text, and it uses the same map (Figure 5-6), giving you 48 rows of 80 blocks.

The IOUDis (\$C07E) switch must be on to allow you to use locations \$C05E and \$C05F to change DHiRes. The firmware in fact leaves it on—and your program should, too—unless it wants to use locations \$C05E and \$C05F to change mouse values (Chapter 9).

Table 5-8. Display Soft Switches

Name	Action	Hex	Function
AltChar	W	\$C00E	Off: Display text using primary character set
AltChar	W	\$C00F	On: Display text using alternate character set
RdAltChar	R7	\$C01E	Read AltChar switch (1 = on)
80Col	W	\$C00C	Off: Display 40 columns
80Col	W	\$C00D	On: Display 80 columns
Rd80Col	R7	\$C01F	Read 80Col switch (1 = on)
80Store	W	\$C000	Off: Cause Page2 on to select auxiliary RAM
80Store	W	\$C001	On: Allow Page2 to switch main RAM areas
Rd80Store	R7	\$C018	Read 80Store switch (1 = on)
Page2	R/W	\$C054	Off: Select Page 1
Page2	R/W	\$C055	On: Select Page 1X (80Store on) or 2
RdPage2	R7	\$C01C	Read Page2 switch (1 = on)
TEXT	R/W	\$C050	Off: Display graphics or (if MIXED on) mixed
TEXT	R/W	\$C051	On: Display text
RdTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C053	Off: Display only text or only graphics
MIXED	R/W	\$C054	On: (If TEXT off) display text and graphics
RdMIXED	R7	\$C01B	Read MIXED switch (1 = on)

Table 5-8—continued. Display Soft Switches

Name	Action	Hex	Function
HiRes	R/W	\$C057	Off: (If TEXT off) display low-resolution graphics
HiRes	R/W	\$C058	On: (If TEXT off) display high-resolution or (if DHiRes on) double-high-resolution graphics
RdHiRes	R7	\$C01D	Read HiRes switch (1 = on)
IOUDis	W	\$C07E	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch*
IOUDis	W	\$C07F	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*
RdIOUDis	R7	\$C07E	Read IOUDis switch (1 = off)†
DHiRes	R/W	\$C05E	On: (If IOUDis on) turn on double-high-resolution
DHiRes	R/W	\$C05F	Off: (If IOUDis on) turn off double-high-resolution
RdDHiRes	R7	\$C07F	Read DHiRes switch (1 = on)†

* The firmware normally leaves IOUDis on. See also the following footnote.

† Reading or writing any address in the range \$C070–\$C07F also triggers the paddle timer and resets VBLInt (Chapter 9).

Table 5-9. Display Modes Supported by Firmware (Including Applesoft)

Display Col/Res	Type	Page	Switches:		Page2	TEXT	MIXED	HiRes	DHiRes
40-column	text	1	off		off	on	off	off	off
80-column	text	1	on	*		on			
low-res	graphics	1	off		off	off	off	off	off
40/low	mixed	1	off		off	off	on	off	
80/low	mixed	1	on	*	off	off	on	off	off
hi-res	graphics	1	off		off	off	off	on	
hi-res	graphics	2	off		on	off	off	on	
40/high	mixed	1	off		off	off	on	on	
80/high	mixed	1	on	*	off	off	on	on	off

* 80Store is set by the firmware when 80Col is turned on.

Table 5-10. Other Display Modes

Display Col/Res	Type	Page	Switches:		Page2	TEXT	MIXED	HiRes	DHiRes
			80Col	80Store					
40-column	text	2	off		on	on			
80-column		2	on	off	on	on			
low-res	graphics	2	off		on	off	off	off	
40/low	rnixed	2	off		on	off	on	off	
80/low	rnixed	2	on	off	on	off	on	off	off
dbl-low	graphics	1	on	*	off	off	off	off	on
dbl-low	graphics	2	on	off	on	off	off	off	on
80/dbl-low	rnixed	1	on	*	off	off	on	off	on
80/dbl-low	rnixed	2	on	off	on	off	on	off	on
40/high	rnixed	2	off		on	off	on	on	
80/high	rnixed	2	on	off	on	off	on	on	off
dbl-high	graphics	1	on	*	off	off	off	on	on
dbl-high	graphics	2	on	off	on	off	off	on	on
80/dbl-high	rnixed	1	on	*	off	off	on	on	on
80/dbl-high	rnixed	2	on	off	on	off	on	on	on

* 80Store is set by the firmware when 80Col is turned on, and must be turned off to use the second 80-column or double-high-resolution page. This means that you cannot use firmware routines such as COut when displaying Page 2 modes not supported by firmware.

For example, to switch to mixed 80-column and double-high-resolution display Page 1, you can use these instructions in your program:

STA	\$C00D	Turns on 80Col; firmware then turns on 80Store.
LDA	\$C054	Turns off Page2; you could also have done a STA.
STA	\$C050	Turns off TEXT; that is, turns on graphics mode.
STA	\$C053	Turns on MIXED; it works now that TEXT is off.
STA	\$C057	Turns on HiRes; it works now that TEXT is off.
STA	\$C07E	Makes sure IOUDis is on so you can access DHiRes.
LDA	\$C05E	Turns on DHiRes; it works now that IOUDis is on.

5.7 Display Page Maps

You should never have to store directly into display memory. Most high-level languages let you write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should use the display features of the built-in I/O firmware.

▲Warning

Never call any firmware with 80Col on or with 80Store and Page2 both on. If you do, the firmware will not function properly. As a general rule, always leave Page2 off.

All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hex). For example, the first 128-byte block contains the data for rows 0, 8, and 16. The next 128-byte block contains data for rows 1, 9, and 17, and so on.

The display memory maps are shown in Figures 5-5 through 5-9. For a full description of the way the Apple IIc hardware handles display memory, see Section 11.9.2.

High-resolution graphics data are stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters.

The first 1024 bytes of the high-resolution display page contain the first row of dots from *each* of the 24 groups of eight rows of dots. The second 1024 bytes of the high-resolution display page contain the second row of dots from *each* group of eight rows of dots, and so on for all eight rows of all the groups. This fills up the 8192 bytes of the high-resolution display page.

The display maps show addresses only for each Page 1. To obtain addresses for text or low-resolution graphics Page 2, add 1024 (\$0400); to obtain addresses for high-resolution Page 2, add 8192 (\$2000).

The 80-column display works a little differently. Half of the data are stored in the normal text Page 1 memory, and the other half are stored in the *auxiliary* memory text Page 1. The display circuitry fetches bytes from the same address in both memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The characters in the even-numbered columns of the display are stored (starting with column 0) in main memory, and the characters in the odd-numbered columns of the display are stored (starting with column 1) in main memory.

For more details about the way the displays are generated, see Chapter 11.

To store display data in auxiliary memory, first turn on the 80Store soft switch by writing to location \$C001. With 80Store on, the page-select switch Page2 selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the auxiliary memory. To select auxiliary memory, turn the Page2 soft switch on by reading or writing at location \$C055.

The double-high-resolution graphics display stores information in the same way as high-resolution graphics, except there is an auxiliary memory location as well as a main memory location corresponding to each address. The two sets of display information are interleaved in a manner similar to the interleaving of two 40-column displays to create an 80-column text display (Figure 5-9).

Figure 5-5. Map of 40-Column Text Display

			\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F	\$10	\$11	\$12	\$13	\$14	\$15	\$16	\$17	\$18	\$19	\$1A	\$1B	\$1C	\$1D	\$1E	\$1F	\$20	\$21	\$22	\$23	\$24	\$25	\$26	\$27
Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39		
0	\$400	1024																																								
1	\$480	1152																																								
2	\$500	1280																																								
3	\$580	1408																																								
4	\$600	1536																																								
5	\$680	1664																																								
6	\$700	1792																																								
7	\$780	1920																																								
8	\$428	1064																																								
9	\$4A8	1192																																								
10	\$528	1320																																								
11	\$5A8	1448																																								
12	\$628	1576																																								
13	\$6A8	1704																																								
14	\$728	1832																																								
15	\$7A8	1960																																								
16	\$450	1104																																								
17	\$4D0	1232																																								
18	\$550	1360																																								
19	\$5D0	1488																																								
20	\$650	1616																																								
21	\$6D0	1744																																								
22	\$750	1872																																								
23	\$7D0	2000																																								

Figure 5-6. Map of 80-Column Text Display

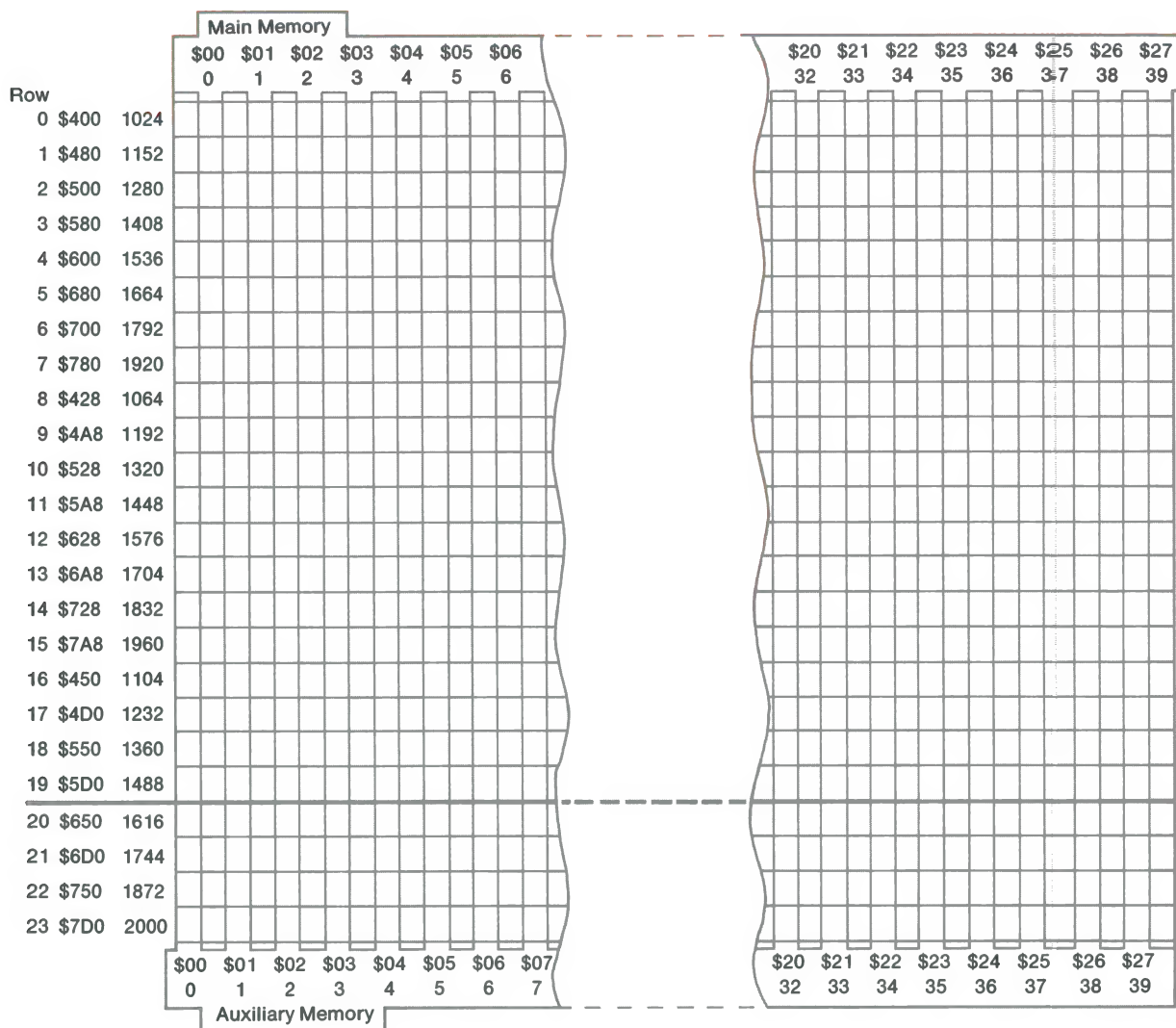


Figure 5-7. Map of Low-Resolution Graphics Display

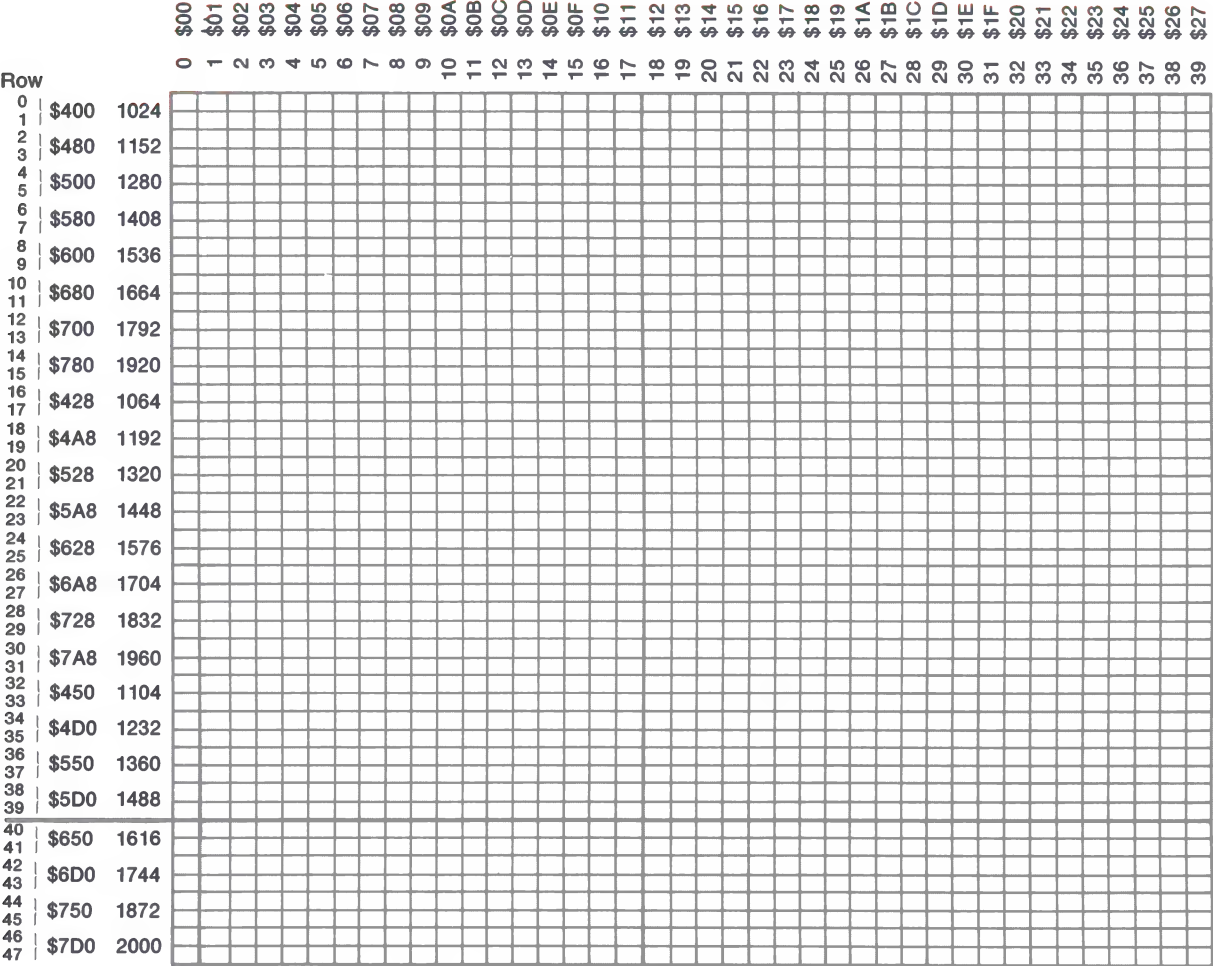


Figure 5-8. Map of High-Resolution Graphics Display

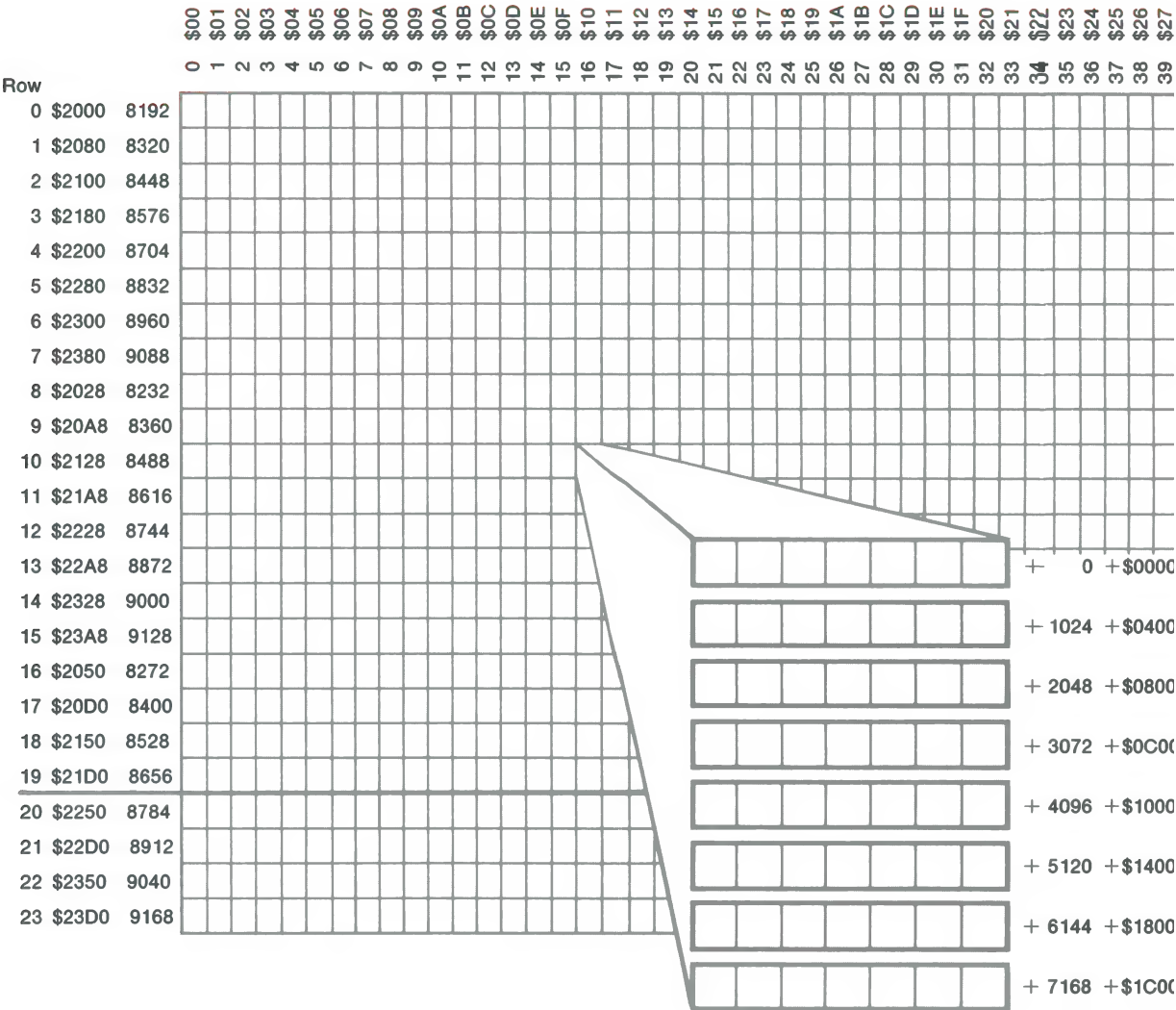
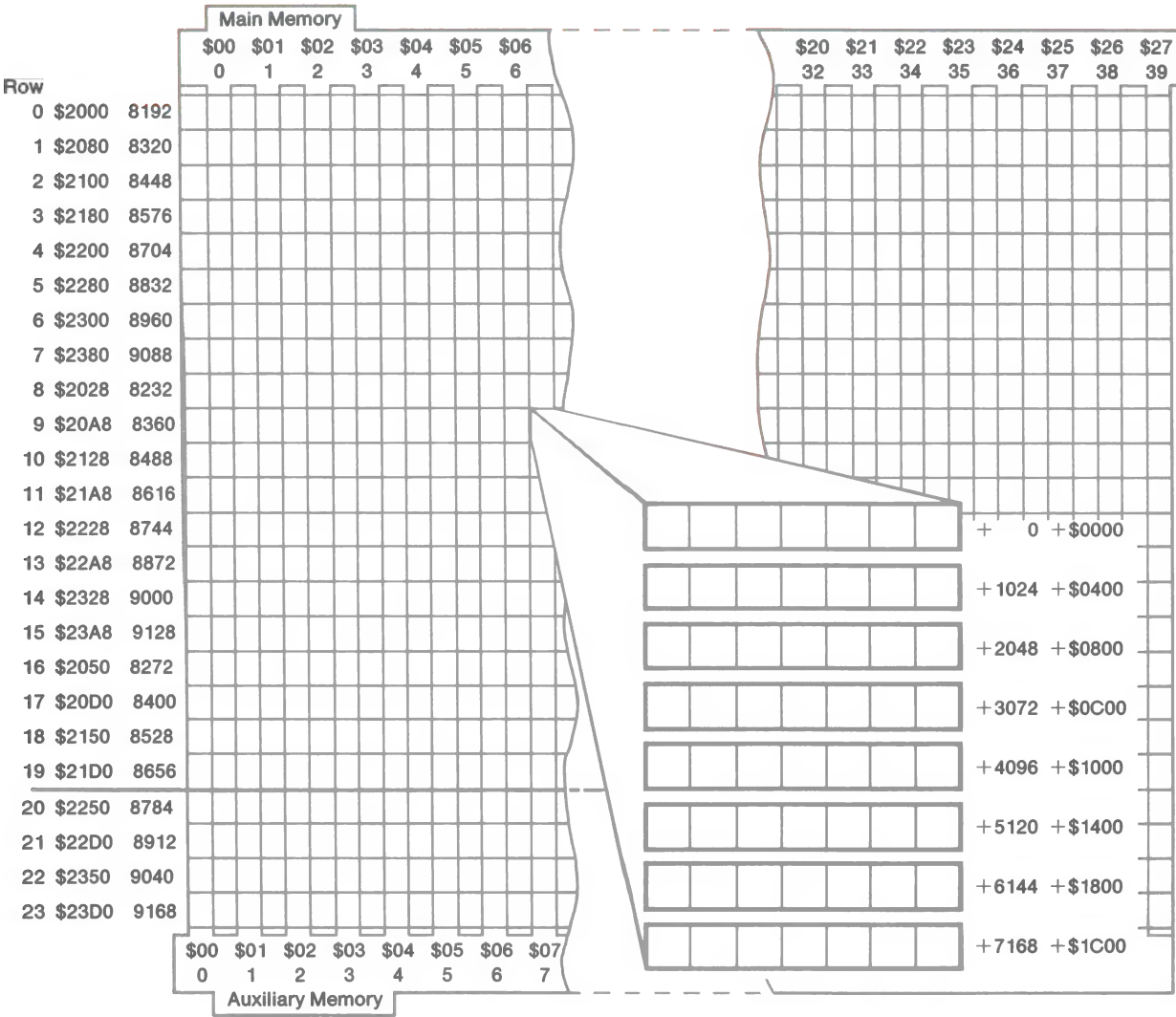


Figure 5-9. Map of Double-High-Resolution Graphics Display



5.8 Monitor Support for Video Display Output

Table 5-11 summarizes the addresses and functions of the video display support routines the Monitor provides. Except for COut and COut1, which are explained in Chapter 3, these routines are described in the subsections that follow.

Table 5-11. Monitor Firmware Routines

Name	Location	Description
ClrEOL	\$FC9C	Clears to end of line from current cursor position
ClEOLZ	\$FC9E	Clears to end of line using contents of Y register as cursor position
ClrEOP	\$FC42	Clears to bottom of window
ClrScr	F832	Clears the low-resolution screen
ClrTop	\$F836	Clears top 40 lines of low-resolution screen
COut	\$FDED	Calls output routine whose address is stored in CSW (normally COut1, Chapter 3)
COut1	\$FDF0	Displays a character on the screen (Chapter 3)
CROut	\$FD8E	Generates a carriage return character
CROut1	\$FD8B	Clears to end of line, then generates a carriage return character
HLine	\$F819	Draws a horizontal line of blocks
HOME	\$FC58	Clears the window and puts cursor in upper-left corner of window
PLOT	\$F800	Plots a single low-resolution block on the screen
PrBl2	\$F94A	Sends 1 to 256 blank spaces to the output device whose address is in CSW
PrByte	\$FDDA	Prints a hexadecimal byte
PrErr	\$FF2D	Sends ERR and CONTROL-G to the output device whose output routine address is in CSW
PrHex	\$FDE3	Prints four bits as a hexadecimal number
PrntAX	\$F941	Prints contents of A and X in hexadecimal

Table 5-11—continued. Monitor Firmware Routines

Name	Location	Description
SCRN	\$F871	Reads color value of a low-resolution block on the screen
SetCol	\$F864	Sets the color for plotting in low-resolution
VTabZ	\$FC24	Sets cursor vertical position (setting CV at location \$25 does not change vertical position until a carriage return)
VLine	\$F828	Draws a vertical line of low-resolution blocks

ClrEOL

ClrEOL clears a text line from the cursor position to the right edge of the window. This routine destroys the contents of A and Y.

CIEOLZ

CIEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL indexed by the contents of the Y register. This routine destroys the contents of A and Y.

ClrEOP

ClrEOP clears the text window from the cursor position to the bottom of the window. This routine destroys the contents of A and Y.

ClrScr

ClrScr clears the low-resolution graphics display to black. If you call this routine while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. This routine destroys the contents of A and Y.

ClrTop

ClrTop is the same as ClrScr, except that it clears only the top 40 rows of the low-resolution display.

COut

COut calls the current character output subroutine (Section 3.3.1). The character to be sent to the output device should be in the accumulator. COut calls the subroutine whose address is stored in CSW (locations \$36 and \$37), usually the standard character output COut1.

COut1

COut1 (Section 3.3.2) displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). It handles these control characters: carriage return, line feed, backspace, and bell. When it returns control to the calling program, all registers are intact.

CROut

CROut sends a carriage return to the current output device.

CROut1

CROut1 clears the screen from the current cursor position to the edge of the text window, then calls CROut.

HLine

HLine draws a horizontal line of blocks of the color set by SetCol on the low-resolution graphics display. Call HLine with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLine returns with A and Y scrambled and X intact.

HOME

HOME clears the display and puts the cursor in the upper-left corner of the screen.

PLOT

PLOT puts a single block of the color value set by SetCol on the low-resolution display screen. Call PLOT with the vertical coordinate of the line in the accumulator, and its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

PrBl2

PrBl2 sends from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to send. If X = \$00, then PrBlank will send 256 blanks.

PrByte

PrByte sends the contents of the accumulator in hexadecimal to the current output device. The contents of the accumulator are scrambled.

PrErr

PrErr sends the word ERR, followed by a bell character (ASCII \$07), to the standard output device. On return, the accumulator is scrambled.

PrHex

PrHex prints the lower nibble of the byte in the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.

PrntAX

PrntAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte printed, and the X register contains the second. On return, the contents of the accumulator are scrambled.

SCRN

SCRN returns the color value of a single block on the low-resolution display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. The block's color is returned in the accumulator. No other registers are changed.

SetCol

SetCol sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 5-4.

VLine

VLine draws a vertical line of blocks of the color set by SetCol on the low-resolution display. Call VLine with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLine returns with the accumulator scrambled.

5.9 I/O Firmware Support for Video Display Output

Apple IIc video firmware conforms to the I/O firmware protocol described in Section 3.4.2. However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode.

The video (port 3) protocol table is shown in Table 5-12.

Table 5-12. Port 3 Firmware Protocol Table

Address	Value	Description
\$C30B	\$01	Generic signature byte of firmware cards
\$C30C	\$88	80-column card device signature
\$C30D	\$ii	\$C3ii is entry point of initialization routine (PInit).
\$C30E	\$rr	\$C3rr is entry point of read routine (PRead).
\$C30F	\$ww	\$C3ww is entry point of write routine (PWrite).
\$C310	\$ss	\$C3ss is entry point of the status routine (PStatus).

PInit

PInit does the following:

- sets a full 80-column window
- sets 80Store (\$C001)
- sets 80Col (\$C00D)
- switches on AltChar (\$C00F)
- clears the screen; places cursor in upper-left corner
- displays the cursor

PRead

PRead reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a 0 in the X register to indicate IOResult = GOOD.

PWrite

PWrite should be called after placing a character in the accumulator with its high bit cleared. PWrite does the following:

- ☐ turns the cursor off
- ☐ if the character in the accumulator is not a control character, turns the high bit on for normal display or off for inverse display, displays it at the current cursor position, and advances the cursor; if at the end of a line, does carriage return but not line feed
- ☐ carries out control functions as shown in Table 5-13

Table 5-13. Pascal Video Control Functions

CONTROL-	Hex	Function
E or e	\$05	Turns cursor on (enables cursor display)
F or f	\$06	Turns cursor off (disables cursor display)
G or g	\$07	Sounds bell (beeps)
H or h	\$08	Moves cursor left one column; if cursor was at beginning of line, moves it to end of previous line
J or j	\$0A	Moves cursor down one row; scrolls if needed
K or k	\$0B	Clears to end of screen
L or l	\$0C	Clears screen; moves cursor to upper-left of screen
M or m	\$0D	Moves cursor to column 0
N or n	\$0E	Displays subsequent characters in normal video; characters already on display are unaffected
O or o	\$0F	Displays subsequent characters in inverse video; characters already on display are unaffected
V or v	\$16	Scrolls screen up one line; clears bottom line
W or w	\$17	Scrolls screen down one line; clears top line
Y or y	\$19	Moves cursor to upper-left (home) position on screen

Table 5-13—continued. Pascal Video Control Functions

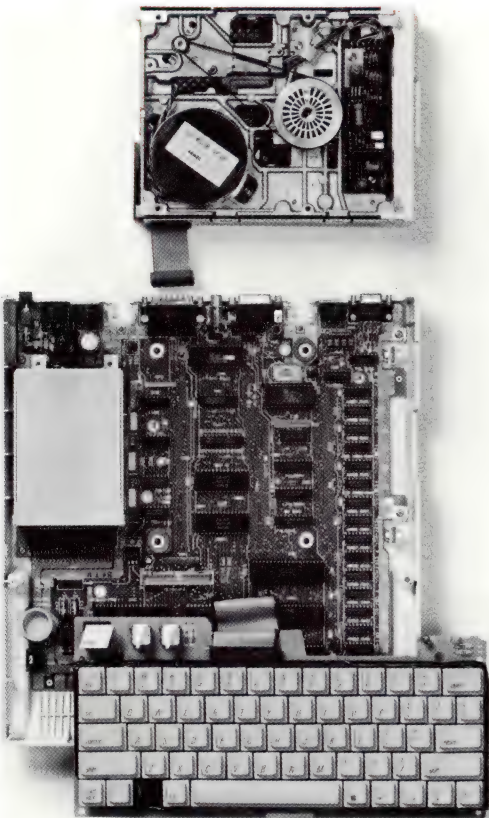
CONTROL-	Hex	Function
Z or z	\$1A	Clears entire line that cursor is on
or \	\$1C	Moves cursor right one column; if at end of line, does CONTROL-M
} or]	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor
^ or 6	\$1E	GOTOxy: Initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively
—	\$1F	If not at top of screen, moves cursor up one line

When PWrite has completed this, it

- turns the cursor back on (if it was not intentionally turned off)
- puts a 0 in the X register (IOResult = GOOD) and returns to the calling program.

PStatus

A program that calls PStatus must first put a request code in the accumulator: either a 0 (meaning “Ready for output?”) or a 1 (meaning “Is there any input?”). PStatus returns with the reply in the carry bit: 0 (no) or 1 (yes). If the request was not 0 or 1, PStatus returns with a 3 in the X register (IOResult = ILLEGAL OPERATION); otherwise, PStatus returns with a 0 in the X register (IOResult = GOOD).



The Apple IIc supports both its built-in disk drive and an optional external 5¼-inch drive; both drives use single-sided, 143K-capacity, 35-track, 16-sector format. Table 6-1 summarizes disk I/O port characteristics.

UniDisk 3.5

A new version of the Apple IIc supports UniDisk 3.5, a 3½-inch, double-sided, 800K-capacity disk drive. The UniDisk 3.5 is an *intelligent* device: the Apple IIc does not have to do all the work of controlling it when reading or writing information on the disk. Programs make requests through the operating system to the intelligent device, the device works on the request, and, when finished, responds to the operating system.

The Protocol Converter in the firmware of the new version of the Apple IIc, described in this chapter, provides a way of communicating with UniDisk 3.5. The Protocol Converter communicates through the Apple IIc's disk I/O port and allows intelligent devices to be daisy chained, which means that the first device is plugged into the disk port, the next device is plugged into the first device, and so on.

The Protocol Converter can handle up to 127 devices in a daisy chain, but the Apple IIc sets a much lower limit, since it can provide power to only a few devices at one time. The Disk IIc is still supported, as long as it is the last device in the daisy chain.

Table 6-1. Disk I/O Port Characteristics

Port number:	I/O port 6 drive 1 (built-in drive) I/O port 6 drive 2 (external drive)
Commands:	IN#6 or PR#6 CALL -151 (to get to the Monitor from BASIC), then 6 CONTROL-K or 6 CONTROL-P
Initial characteristics:	All resets except CONTROL - RESET with a valid reset vector eventually pass control to the built-in disk drive.
Hardware location:	
\$C0E0-EF	Reserved
Monitor firmware routines:	None
I/O firmware entry points:	\$C600 (port 6)
Use of screen holes:	Port 6 main and auxiliary memory screen holes are reserved.

The external disk drive connector is described in Section 11.10.

The disk I/O firmware resides in the \$C600 address space. It supports the built-in drive as if it were slot 6 drive 1, and the external drive as if it were slot 6 drive 2. If disk startup is unsuccessful, the firmware shuts off the disk drive motor and displays the message **Check Disk Drive** on the display screen.


UniDisk 3.5

For Apple IIc's that support UniDisk 3.5, disk startup is tried with both the built-in drive and the first device connected to the disk I/O port. If both tries fail, the message **Check Disk Drive** appears on the display screen.

6.1 Startup

The Apple IIc has two ways to start up—a cold start and a warm start. A cold start clears the machine's memory and tries to load an operating system from disk. A warm start stops the current program that is running and leaves the machine in Applesoft with memory and programs intact.

A cold start can be initiated by any of the following:

- ☐ turning the machine on
- ☐ pressing -**CONTROL**-**RESET**
- ☐ issuing a reboot command from the Monitor, BASIC, or a program
- ☐ pressing **CONTROL**-**RESET**, if a valid reset vector does not exist

The cold-start routine first sets a number of soft switches (see Chapter 2) and then passes control to the program entry point at \$C600. This code turns on the internal drive motor, recalibrates the read/write head at track 0, then reads sector 0 from that track. The sector contents are loaded into memory starting at address \$0800; then program control passes to \$0801. The program loaded depends on the operating system or application program on the disk.

To restart the system from BASIC, issue a PR#6 command; from Monitor command mode, issue 6 CONTROL-P; and use JMP \$C600 from a machine-language program.

A warm start begins when you press **CONTROL**-**RESET**, if a valid reset vector exists. Normally, a warm start leaves you in BASIC with memory unchanged. If a program has changed the reset vector the system won't do a warm start; instead, it may do any number of things. Usually it either does a cold start or it beeps or does nothing, leaving you in the currently executing program.

6.2 External Drive Startup

You might need to start your Apple IIc from an external drive if, for example, the built-in drive fails. The way this works depends on the type of Apple IIc you have and the software you want to run.

Original IIc

The ProDOS operating system (but not the DOS or Pascal operating systems) supports startup using the external Disk IIc drive. This ProDOS feature makes it possible to start the Apple IIc with a diagnostic program in the event that the built-in drive does not work.

To restart from the Monitor using the external Disk IIc, insert a ProDOS disk in the external drive, then start the Monitor by typing `CALL -151` and pressing `[7] [CONTROL] [P]`.

To restart from BASIC using the external Disk IIc, insert a ProDOS disk in the external drive, then type `PR#7`.

External drive startup using PR#7 works on the original Apple IIc only with ProDOS-based programs. It does not work with Pascal 1.0 or later versions, or with DOS 3.3.

UniDisk 3.5

The Apple IIc that supports UniDisk 3.5 can use PR#5 to start up from an external Disk IIc or from the first intelligent drive connected to the Protocol Converter through the disk port. Starting up from an external intelligent drive works from ProDOS or Pascal 1.3, but not from DOS or versions of Pascal earlier than 1.3.

6.3 The Protocol Converter

UniDisk 3.5

The rest of this chapter applies only to the version of the Apple IIc that supports UniDisk 3.5.

The rest of this chapter is about the Protocol Converter, which is a set of assembly-language routines used to support external I/O devices, such as UniDisk 3.5. To ProDOS and Pascal 1.3, the Protocol Converter appears to be a block device.

The following topics are discussed:

- how to locate the Protocol Converter
- how to issue a call to the Protocol Converter
- how to use each call
- the parameters required for each call
- possible error codes returned for each call
- the possible causes of the errors

A **block device** reads or writes information in organized groups called blocks, which are typically 512 bytes in size. A disk drive is a block device. Compare this with a **character device**, which reads or writes data sequentially. Printers, modems, and plotters are character devices.

At the end of this chapter is an example of an assembly-language program that uses a Protocol Converter call.

6.3.1 Locating the Protocol Converter

The Protocol Converter code in the Apple IIc's firmware always begins at address \$C500. To ensure compatibility of your programs with the Apple IIe, however, your Protocol Converter routines should always begin with a search for the Protocol Converter. Your program can identify the Protocol Converter by finding the following bytes:

```
$Cn01 = $20
$Cn03 = $00
$Cn05 = $03
$Cn07 = $00
```

where n can be an integer from 1 to 7. The Protocol Converter entry point is then found at address $\$Cn00 + (\$CnFF) + 3$, where $(\$CnFF)$ refers to the value of the byte located at $\$CnFF$. The sample program at the end of this chapter illustrates such a search.

6.3.2 Issuing a Call to the Protocol Converter

MLI calls: See the *ProDOS Technical Reference Manual*, Chapter 4.

Protocol Converter calls are coded like ProDOS Machine Language Interface (MLI) calls: the program executes a JSR to a dispatch routine at address $\$C500 + (\$C5FF) + 3$, where $(\$C5FF)$ refers to the value of the byte located at $\$C5FF$.

The Protocol Converter call number and a two-byte pointer to the call's parameter list must immediately follow the call. Here is an example of a call to the Protocol Converter:

IWMCALL	JSR	DISPATCH	Calls PC command dispatcher
	DFB	CmdNum	Specifies the command type
	DW	CmdList	2-byte (low, high) pointer to parameter list
	BCS	ERROR	Sets carry on an error

The command number (CmdNum) defines which Protocol Converter call you want to make. Most Protocol Converter calls include a two-byte pointer to a parameter list. The parameter list can contain information to be used by the call, or can provide space for information to be returned by the call. The length and content of the parameter list depend on the call being made. The format of each Protocol Converter call's parameter list is described in Section 6.4.

When the call has finished, the program resumes execution at the statement following the pointer to the parameter list. In the example above, the DFB and DW statements are skipped, and execution resumes with the BCS statement. If the call is successful, the C flag (in the processor status register) is cleared (0), and the accumulator (the A register) is cleared to all 0s. If the call is unsuccessful, the C flag is set (1), and the error code is placed in the A register. After the Protocol Converter call, the contents of the 65C02's registers are as follows:

Register:	Processor Status							X	Y	A	PC	S
Bit:	N	Z	C	D	V	I	B					
Successful call:	x	x	0	0	x	u	u	x	x	0	JSR+3	u
Unsuccessful call:	x	x	1	0	x	u	u	x	x	Error	JSR+3	u

x = undefined, except in cases where index information is returned in X and Y
u = unchanged

6.3.3 Cautions

You *must* observe the following cautions when using the Protocol Converter, or *your program will crash*:

- Leave space on the stack for the Protocol Converter. The Protocol Converter requires up to 35 bytes of stack space. Be sure you take this into account when calculating the stack space used by your program. If you don't do this, you program will fail if it tries to access data that *used* to be on the stack.
- Be sure that all RAM that you intend the Protocol Converter to access is both read-enabled and write-enabled. The Protocol Converter must be able to read from the RAM after writing to it, to obtain a checksum. Failure to observe this rule results in an error (BusErr \$06).
- Don't pass data to or from the Protocol Converter through any zero page locations. Some of these locations are reserved for temporary storage of data by the Protocol Converter, and your data will get changed.

Reading and writing to RAM: See Section 2.4.

6.4 Descriptions of the Protocol Converter Calls

Calls to the Protocol Converter are used

- to obtain status information about a device
- to reset a device
- to format the medium in a device
- to read from a device
- to write to a device
- to send control information to a device.

The Protocol Converter calls, in command-number sequence, are:

STATUS (\$00)	Returns status information about a particular device, including general status (character or block device, read or write protection, format allowed, device on line); the device control block (set with the CONTROL call); the device newline status (character devices only); and device-specific information (number of blocks, ID string, device name, device type, device firmware version).
READ BLOCK (\$01)	Reads one 512-byte block from a disk device, and writes it to memory.
WRITE BLOCK (\$02)	Writes one 512-byte block from memory to a disk device.
FORMAT (\$03)	Prepares all blocks on a block device for reading and writing.
CONTROL (\$04)	Controls some device functions, including soft resets, setting the device control block (which controls global aspects of the device's operating environment), setting newline status (character devices only), and device interrupts. Several CONTROL calls are device-specific.
INIT (\$05)	Resets all resident devices. A global reset is done automatically on startup or system resets from the keyboard; an application should never have to reset all devices.
OPEN (\$06)	Prepares a character device for reading or writing.

CLOSE (\$07)	Tells a character device that a sequence of reads or writes is over.
READ (\$08)	Reads a specified number of bytes from a specified device.
WRITE (\$09)	Writes a specified number of bytes from memory to a specified device.

The following sections describe each Protocol Converter call, including the command number, the parameter list, and error codes. The calls are discussed in command-number order in this format:

Command Name: The name used to identify the call.

Command Number: A hexadecimal number that specifies which call is being made to the Protocol Converter.

Parameter List: A list of required call parameters.

General Description: What the call does and what you use it for.

Parameter Descriptions: A description of each parameter and the data it refers to. When a parameter refers to a status or control code, the meaning of each code number is discussed.

Possible Errors: A list of the error codes that can be returned by this call. A complete list of Protocol Converter error codes is included at the end of this chapter.

6.4.1 STATUS

Command Number	\$00
Parameter List	\$03 (parameter count) Unit number Status list pointer (low byte, high byte) Status code

The STATUS call returns status information about a specified device. The type of information returned is determined by the device and its status-code parameter. The status list pointer defines where the status information is returned to.

STATUS returns the number of bytes of status information that it generates in the X and Y registers, the low byte of this number in the X register, and the high byte in the Y register.

Parameter Descriptions

Parameter Count

1-byte value 3 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the chain.

Important!

You can get the status of the Protocol Converter itself if you use a unit number of \$00 and a status code of \$00 in a STATUS call (see the discussion beginning "Status code = \$00," below).

Status List Pointer

2-byte value Points to the buffer to which the status is to be returned. The length required for the buffer varies depending on the status request being made.

Status Code 1-byte value	Indicates what kind of status request is being made. Status codes are in the range \$00-\$FF, as follows:
	Code Status Returned
	\$00 Return device status
	\$01 Return device control block (DCB) (not supported by UniDisk 3.5)
	\$02 Return newline status (character devices only) (not supported by UniDisk 3.5)
	\$03 Return device information block (DIB)
	\$05 Return UniDisk 3.5 status

Status code = \$00 returns a device status consisting of four bytes. The first is the general status byte, with the following format:

Bit	Description
7	0 = character device, 1 = block device
6	1 = write allowed
5	1 = read allowed
4	1 = device on line or disk in drive
3	0 = format allowed
2	0 = medium write protected (block devices only)
1	1 = device currently interrupting
0	1 = device currently open (character devices only)

If the STATUS call is for a block device, the next three bytes (low byte first) are the size in 512-byte blocks. The maximum size is 16 million (\$FFFFFF) blocks (about 8 gigabytes). If the call is for a character device, these three bytes must be set to 0.

A STATUS call with status code = \$00 and unit number = \$00 returns the status of the Protocol Converter itself. In this case, the status list consists of eight bytes, as follows:

STAT_LIST	DFB	Number_Devices	Devices hooked to PC
	DFB	Interrupt_Status	Bit 6 clear = interrupt sent
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved

The `Number__Devices` byte returns the total number of intelligent devices attached to the Protocol Converter. The `Interrupt__Status` byte is a copy of the asynchronous communications interface adapter (ACIA) status register at the time of the interrupt, and is used to indicate that a device requires interrupt servicing. If the sixth bit of this byte equals 0, one or more devices in the Protocol Converter bus daisy chain must be serviced; your interrupt handler must poll each device on the chain to determine which ones.

About Interrupts: Devices that require interrupt servicing must use the EXTINT line on the Apple IIc's external disk port connector to be supported by the Protocol Converter.

For example, UniDisk 3.5 does not support this line, and so cannot generate interrupts to the Protocol Converter. See Section 6.4.5 for instructions on enabling Protocol Converter interrupts. See Appendix E for more information about programming with interrupts.

Status code = \$01 returns the device control block (DCB). The DCB is used to control various operating characteristics of a device, and is device dependent. Each device has a default DCB, which can be altered with a CONTROL call. The first byte (the count byte) gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 256 bytes (257 including the count byte). Note that UniDisk 3.5 has no DCB, and returns an error (BadCtl \$21) in response to this call.

Status code = \$02 returns newline status. Newline status applies only to character devices. A status code = \$02 passed to a block device returns a BadCtl (\$21) error.

Status code = \$03 returns the device information block (DIB). The device's information block identifies the device, its type, and various other attributes. The returned status list has the following form:

STAT_LIST	DFB	Device_Statbyte1	Same as byte 1 in status code = 0
	DFB	Device_Size_Lo	Number of blocks (block device)
	DFB	Device_Size_Med	Number of blocks (middle byte)
	DFB	Device_Size_Hi	Number of blocks (high byte)
	DFB	ID_String_Length	Length in bytes (16 max.)
	ASC	'<device name>'	7-bit ASCII, uppercase, padded with spaces, 8th bit always=0 (16 bytes)
	DFB	Device_Type_Code	
	DFB	Device_Subtype_Code	
	DW	Version	Device firmware version number

Newline read mode: See Chapter 4 in the *ProDOS Technical Reference Manual*.

Status code = \$05 returns the UniDisk 3.5 status. This call allows a diagnostic program to get more detailed information about the cause of a read or write error, and to examine the contents of the 65C02's registers after a CONTROL call with control code = \$05 (see Section 6.4.5). The returned status list has this form:

STAT_LIST	DFB	\$00	
	DFB	Error	Soft Error byte (see below)
	DFB	Retries	Number of retries (see below)
	DFB	\$00	
	DFB	A_Value	Acc value after a CONTROL EXECUTE call
	DFB	X_Value	X value after EXECUTE
	DFB	Y_Value	Y value after EXECUTE
	DFB	P_Value	Processor status value after EXECUTE

The Error byte returned by a STATUS call with status code = \$05 contains the following bits:

Bit	Description
7	0
6	0
5	1 = address field mark or checksum error
4	1 = data field checksum error
3	1 = data field bit-slip mark mismatch
2	1 = seek error; unexpected track value found in address field
1	0
0	0

The Retries byte returned by a STATUS call with status code = \$05 specifies the number of address fields that had to be passed before the operation was completed. This information could be used, for example, to determine the number of passes necessary to read a data field correctly: If Retries is found to be greater than the number of sectors on the target track, then more than one pass was required.

The last four bytes of the status list are set only after a CONTROL call with control code = \$05, and are 0 after any other call (STATUS calls do not clear the status bytes).

Possible Errors

The following errors can be returned by the STATUS call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$21	BadCtl	Invalid status code
\$30-\$3F		Device-specific errors

6.4.2 READ BLOCK

Command Number \$01

Parameter List \$03 (parameter count)
Unit number
Data buffer (low byte, high byte)
Block number (low byte, mid byte, high byte)

The READ BLOCK call reads one 512-byte block into memory from the block device specified by the unit-number parameter. The block of data is placed in a buffer starting at the address specified by the data-buffer parameter.

Parameter Descriptions

Parameter Count

1-byte value 3 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Data Buffer 2-byte value	Points to the buffer into which the data are read. The buffer must be 512 or more bytes in length.
Block Number 3-byte value	The logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

Possible Errors

The following errors can be returned by the READ BLOCK call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

6.4.3 WRITE BLOCK

Command Number	\$02
Parameter List	\$03 (parameter count) Unit number Data buffer (low byte, high byte) Block number (low byte, mid byte, high byte)

The WRITE BLOCK call writes one 512-byte block from memory to the disk device specified by the unit-number parameter. The block in memory starts at the address specified by the data-buffer parameter.

Parameter Descriptions

Parameter

Count

1-byte value 3 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Data Buffer

2-byte value Points to the buffer from which the data are to be written.

Block Number

3-byte value The logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

Possible Errors

The following errors can be returned by the WRITE BLOCK call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2B	NoWrite	Disk write protected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

6.4.4 FORMAT

Command Number	\$03
Parameter List	\$01 (parameter count) Unit number

The FORMAT call prepares all blocks on the recording medium of a block device for reading and writing. The formatting done by this call is specific to each device and is not linked to any operating system; for example, bitmaps and catalogs are not written by this call.

Parameter Descriptions

Parameter Count

1-byte value 1 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the ST ATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the FORMAT call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2B	NoWrite	Disk write protected
\$2F	OffLine	Device off-line or no disk in drive

6.4.5 CONTROL

Command Number	\$04
Parameter List	\$03 (parameter count) Unit number Control list (low byte, high byte) Control code

The CONTROL call sends control information to the device. The information can be of a general nature (such as resets or interrupts), or device-specific (such as Download to UniDisk 3.5 RAM).

Important!

A CONTROL call to unit number \$00 sends control information to the Protocol Converter itself. See the discussions of control code = \$00 and control code = \$01, below.

Parameter Descriptions

Parameter Count

1-byte value 3 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter. Use a unit number of \$00 in the CONTROL call to send control information to the Protocol Converter itself.

Control List

2-byte value Points to the buffer containing the control information. The first two bytes (the count bytes, low byte first) of the control list specify the number of bytes in the list (*not* including the count bytes); the remainder of the list contains the control information passed to the device.

Important!

Every CONTROL call must have a control list; if no control information is being passed, then the control list consists of the count bytes only:

```
CTRL_LIST DW $00
```


Control Code
1-byte value

The number of the control request being made. Control codes are in the range \$00-\$FF. The following requests are not device specific:

Code	Control Function
\$00	Reset the device
\$01	Set device control block (DCB)
\$02	Set newline status (character devices only)
\$03	Service device interrupt

Control requests to unit number \$00 are sent to the Protocol Converter itself:

Code	Control Function
\$00	Enable interrupts from Protocol Converter
\$01	Disable interrupts from Protocol Converter

Specific devices may respond to some or all of these additional control requests:

Code	Control Function
\$04	Eject disk
\$05	Run a 65C02 subroutine
\$06	Set download address
\$07	Download to device RAM

Control code = \$00 performs a warm reset of the device and generally returns “housekeeping” values to some reset value. The control list for this call is device dependent.

The control list for this call for UniDisk 3.5 devices is:

CTRL_LIST DW \$00 No parameters are passed.

A CONTROL call with control code = \$00 and unit number = \$00 enables interrupts from the Protocol Converter. This informs the firmware that external interrupts are possible, and directs it to call the user's interrupt handler if an interrupt occurs. It also turns on the ACIA for port 1.

When the user's interrupt handler identifies an external interrupt, you can determine if it came from the Protocol Converter by making a STATUS call with unit number = \$00 and control code = \$00 (see Section 6.4.1). See Appendix E for more information on handling interrupts.

Control code = \$01 alters the contents of the device control block (DCB). The DCB is used to set global aspects of a device's operating environment. Each device has a default setting for the DCB, set on initialization. Since the length of the DCB is device dependent, you should first read in the DCB with the STATUS call, then alter the bits of interest, and finally, use the same byte string as the control block for the CONTROL call. The first byte (the count byte) of the DCB gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 257 bytes, including the count byte.

Note that because UniDisk 3.5 has no DCB, a Set DCB CONTROL call to UniDisk 3.5 returns an error (BadCtl \$21).

A CONTROL call with control code = \$01 and unit number = \$00 disables interrupts from the Protocol Converter. This call turns off the ACIA for port 1 and sets the least significant bit of the ACIA control register to 0.

Control code = \$02 sets a character device to newline enabled or newline disabled.

Control code = \$03 sends a device service interrupt. This code is to be used as needed for interrupt-driven devices.

Control code = \$04 ejects a disk. This code is to be used for devices that support an auto-eject feature. This code causes UniDisk 3.5 to auto-eject a disk. There are no parameters in the control list, and no errors are returned if the disk ejected correctly or there was no disk in the drive. Error code \$27 (IOError) is returned if the eject failed—that is, if a disk is still in the drive. The control list for UniDisk 3.5 is the following:

CTRL_LIST DW \$00 No parameters are passed.

▲Warning | Control codes \$05 and higher are reserved; use of some of these codes can cause your system to crash.

Possible Errors

The following errors can be returned by the CONTROL call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$21	BadCtl	Invalid control code
\$22	BadCtlParm	Invalid parameter list
\$30-\$3F		Device-specific errors

Newline read mode: See Chapter 4 in the *ProDOS Technical Reference Manual*.

6.4.6 INIT

Command Number	\$05
Parameter List	\$01 (parameter count) \$00 (unit number)

The INIT call resets all intelligent devices attached to the Protocol Converter. The Protocol Converter goes through an initialization sequence, cold-resetting all devices and sending each its unit number. This call is made automatically on startup; an application should never have to make this call.

Parameter Descriptions

Parameter Count

1-byte value 1 for this call.

Unit Number

1-byte value The unit number used in this call is always \$00.

Possible Errors

The following errors can be returned by the INIT call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected

6.4.7 OPEN

Command Number	\$06
Parameter List	\$01 (parameter count) Unit number

The OPEN call prepares a character device for reading or writing.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use an OPEN call with UniDisk 3.5 will result in an error (BadCmd \$01).

Parameter Descriptions

Parameter Count	
1-byte value	1 for this call.
Unit Number	
1-byte value	The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the OPEN call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected
\$2F	OffLine	Device off-line or no disk in drive

6.4.8 CLOSE

Command Number	\$07
Parameter List	\$01 (parameter count) Unit number

The CLOSE call tells a character device that a sequence of reads or writes is over.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use a CLOSE call with UniDisk 3.5 will result in an error (BadCmd \$01).

Parameter Descriptions

Parameter Count	
1-byte value	1 for this call.
Unit Number	
1-byte value	The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the CLOSE call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected
\$2F	OffLine	Device off-line or no disk in drive

6.4.9 READ

Command Number	\$08
Parameter List	\$04 (parameter count) Unit number Buffer pointer (low byte, high byte) Byte count (low byte, high byte) Address pointer (low byte, mid byte, high byte)

The READ call reads into memory the number of bytes specified by the byte-count parameter. The bytes are placed in a buffer starting at the address specified by the buffer-pointer parameter.

Parameter Descriptions

Parameter Count	
1-byte value	4 for this call.
Unit Number	
1-byte value	The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.
Buffer Pointer	
2-byte value	Points to the buffer into which the data is read. The buffer must be large enough to contain the number of bytes requested by the byte-count parameter.
Byte Count	
2-byte value	Specifies the number of bytes to be transferred.
Address Pointer	
3-byte value	Specifies the address to start reading from. The meaning of this parameter depends on the device being read.

Possible Errors

The following errors can be returned by the READ call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

6.4.10 WRITE

Command Number	\$09
Parameter List	\$04 (parameter count) Unit number Buffer pointer (low byte, high byte) Byte count (low byte, high byte) Address pointer (low byte, mid byte, high byte)

The WRITE call writes from memory the number of bytes specified by the byte-count parameter to the specified unit. The bytes in memory start at the address indicated by the buffer-pointer parameter. The meaning of the address pointer depends on the type of device (see parameter descriptions).

Parameter Descriptions

Parameter Count

1-byte value 4 for this call.

Unit Number

1-byte value The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E, and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Buffer Pointer 2-byte value	Points to the buffer from which the data is to be written.
Byte Count 2-byte value	Specifies the number of bytes to be transferred.
Address Pointer 3-byte value	Specifies the address to start writing from. The meaning of this parameter depends on the device being written to.

Possible Errors

The following errors can be returned by the WRITE call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

6.5 An Example: Issuing a Protocol Converter Call

Here is an example of a program that issues a STATUS call to the Protocol Converter to obtain information about a device.

The code for the Protocol Converter in the version of the Apple IIc that supports UniDisk 3.5 always begins at address \$C500; however, to ensure compatibility with the Apple IIe, your programs should always do a search for the Protocol Converter, as in this example.

```
0000:          1 *
0000:          2 *
0000:          3 *
0000:          4 * This example shows how to find
0000:          5 * and use a PC interface. A search
0000:          6 * is made for a PC, and when one is
0000:          7 * found, a vector is set up which
0000:          8 * points to the PC entry. Then a
0000:          9 * Device Information Block STATUS call
0000:         10 * is made, and if successful, the name
0000:         11 * string embedded in the DIB is output
0000:         12 * to the screen. Only the first device
0000:         13 * in the chain is accessed.
0000:         14 *
0000:         15 *
0000:         16             MSB    ON
0000:         17 *
0000:         18 *
0000:    0006  19 ZPTempl equ    $0006      ;Temporary zero
0000:         20 *                      page storage
0000:    0007  21 ZPTempH equ    $0007
0000:         22 *
0000:    FDED  23 COut     equ    $FDED      ;Console output
0000:    FD8E  24 CROut   equ    $FD8E      ;Carriage return
0000:         25 *
0000:    0000  26 StatusCmd equ    0
0000:         27 *
0000:         28 *
0300:    0300  29             org    $300
0300:         30 *
0300:         31 * Find a Protocol Converter in one of the
0300:         32 * slots.
0300:         33 *
0300:20 43 03  34             jsr    FindPC
0303:B0 1C 0321 35             bcs    Error
0305:         36 *
0305:         37 * Now make the DIB call to the first guy
0305:         38 *
0305:20 67 03  39             jsr    Dispatch
```

```

0308:00      40      dfb   StatusCmd
0309:6A 03    41      dw    DParms
030B:B0 14    0321    42      bcs   Error
030D:        43 *
030D:        44 * Got the DIB; now print the name string
030D:        45 *
030D:A2 00    46      ldx   #0
030F:        030F    47 morechars equ   *
030F:BD 74 03 48      lda   DIBName,x
0312:09 80    49      ora   $$80      ;COut wants high
0314:        50 *                      Bit set
0314:        51 *
0314:20 ED FD 52      jsr   COut
0317:E8      53      inx
0318:EC 73 03 54      cpx   DIBNameLen
031B:90 F2    030F    55      blt   morechars
031D:        56 *
031D:20 8E FD 57      jsr   CROut      ;Finish it off
0320:        58 *                      with a return
0320:        59 *
0320:60      60      rts
0321:        61 *
0321:        62 *
0321:        0321    63 Error   equ   *
0321:        64 *
0321:        65 * There's either no PC around, or there
0321:        66 * was no Unit #1... give message
0321:        67 *
0321:A2 00    68      ldx   #0
0323:        0323    69 err1   equ   *
0323:BD 2F 03 70      lda   Message,x
0326:F0 06    032E    71      beq   errout
0328:20 ED FD 72      jsr   COut
032B:E8      73      inx
032C:D0 F5    0323    74      bne   err1
032E:        75 *
032E:        032E    76 errout   equ   *
032E:60      77      rts
032F:        78 *
032F:CE CF A0 D0 79 Message asc   'NO PC OR NO DEVICE'
0341:8D 00    80      dfb   $8D,0
0343:        81 *
0343:        82 *
0343:        0343    83 FindPC   equ   *
0343:        84 *
0343:        85 * Search slot 7 to slot 1 looking for
0343:        86 * signature bytes
0343:        87 *
0343:A2 07    88      ldx   #7      ;Do for seven
0345:        89 *                      slots
0345:A9 C7    90      lda   $$C7
0347:85 07    91      sta   ZPTempH

```

```

0349:A9 00      92      lda    #$00
034B:85 06      93      sta    ZPTempl
034D:           94      *
034D:           95  newslot equ    *
034D:A0 07      96      ldy    #7
034F:           97      *
034F:           98  again   equ    *
034F:B1 06      99      lda    (ZPTempl),y
0351:D9 70 03   100     cmp    sigtab,y      ;One of four
0354:           101      *                  byte signature
0354:F0 07      102     beq    maybe          ;Found one
0356:           103      *                  signature byte
0356:C6 07      104     dec    ZPTempH
0358:CA         105     dex
0359:D0 F2 034D 106     bne    newslot
035B:           107      *
035B:           108      * If we get here, it's because we couldn't
035B:           109      * find a Protocol Converter.
035B:           110      * Exit with the carry set.
035B:           111      *
035B:38         112     sec
035C:60         113     rts
035D:           114      *
035D:           115      * If we get here, it means that one or
035D:           116      * more of the signature bytes
035D:           117      * for this card are what we're looking
035D:           118      * for. Decrement the byte
035D:           119      * counter and branch back to verify any
035D:           120      * remaining bytes.
035D:           121      *
035D:           122  maybe   equ    *
035D:88         123     dey
035E:88         124     dey                  ;If N=1 then
035F:           125      *                  all sig bytes okay
035F:10 EE 034F 126     bpl    again
0361:           127      *
0361:           128      * Found a Protocol Converter interface.
0361:           129      * Set up the call address.
0361:           130      * We already have the high byte ($CN);
0361:           131      * we just need the low byte.
0361:           132      *
0361:           133  foundPC equ    *
0361:A9 FF      134     lda    #$FF
0363:85 06      135     sta    ZPTempl
0365:A0 00      136     ldy    #0                  ;For
0367:           137      *                  indirect load
0367:B1 06      138     lda    (ZPTempl),y      ;Get the
0369:           139      *                  byte
0369:           140      *
0369:           141      * Now the Acc has the low order ProDOS
0369:           142      * entry point. The PC entry is
0369:           143      * three locations past this...

```

```

0369:          144 *
0369:18        145         clc
036A:69 03     146         adc    #3
036C:85 06     147         sta    ZPTempl
036E:          148 *
036E:          149 * Now ZPTempl has the PC entry point.
036E:          150 * Return with carry clear.
036E:          151 *
036E:18        152         clc
036F:60        153         rts
0370:          154 *
0370:          155 *
0370:          156 * These are the PC signature bytes in
0370:          157 * their relative order.
0370:          158 * The $FF bytes are filler bytes and
0370:          159 * are not compared.
0370:          160 *
0370:FF 20 FF 00 161 sigtab    dfb    $FF,$20,$FF,$00
0374:FF 03 FF 00 162            dfb    $FF,$03,$FF,$00
0378:          163 *
0378:          164 *
0378:          165 Dispatch    equ    *
0378:6C 06 00    166            jmp    (ZPTempl)    ;Simulate
037B:          167 *                    an indirect JSR to PC
037B:          168 *
037B:          169 *
037B:          170 DParms      equ    *
037B:03          171 DPParmCt   dfb    3                ;Status
037C:          172 *                    calls have three parameters
037C:01          173 DPUnit      dfb    1
037D:80 03     174 DPBuffer    dw    DIB
037F:03        175 DPStatCode dfb    3
0380:          176 *
0380:          177 *
0380:          178 DIB          equ    *
0380:00          179 DIBStatByte1 dfb    0
0381:00 00 00   180 DIBDevSize dfb    0,0,0
0384:00        181 DIBNameLen dfb    0
0385:          182 DIBName      ds    16,0
0395:00        183 DIBType      dfb    0
0396:00        184 DIBSubType   dfb    0
0397:00 00     185 DIBVersion   dw    0
0399:          186 *
0399:          187 *

```

6.6 Summary of Commands and Parameters

Following is a summary of Protocol Converter calls. In each case, byte 0 of the command parameter list (CmdLst) specifies the number of parameters in the command list (not including byte 0). Parameters that require more than one byte (the status list pointer, for example) are entered low byte first. The meaning of the address-pointer parameter is device specific. See the sections on the individual calls in this chapter for a discussion of each parameter.

Figure 6-1. Summary of Protocol Converter Calls

Command	STATUS	READBLOCK	WRITEBLOCK	FORMAT	CONTROL
CmdNum	\$00	\$01	\$02	\$03	\$04
CmdList Byte					
0	\$03	\$03	\$03	\$01	\$03
1	Unit Num	Unit Num	Unit Num	Unit Num	Unit Num
2	Stat List Ptr	Buffer Ptr	Buffer Ptr		Ctl List Ptr
3					
4	Stat Code	Block Num	Block Num		Ctl Code
5					
6					

Command	INIT	OPEN	CLOSE	READ	WRITE
CmdNum	\$05	\$06	\$07	\$08	\$09
CmdList Byte					
0	\$01	\$01	\$01	\$04	\$04
1	\$00	Unit Num	Unit Num	Unit Num	Unit Num
2				Buffer Ptr	Buffer Ptr
3					
4				Byte Count	Byte Count
5					
6					
7				Address Ptr	Address Ptr
8					

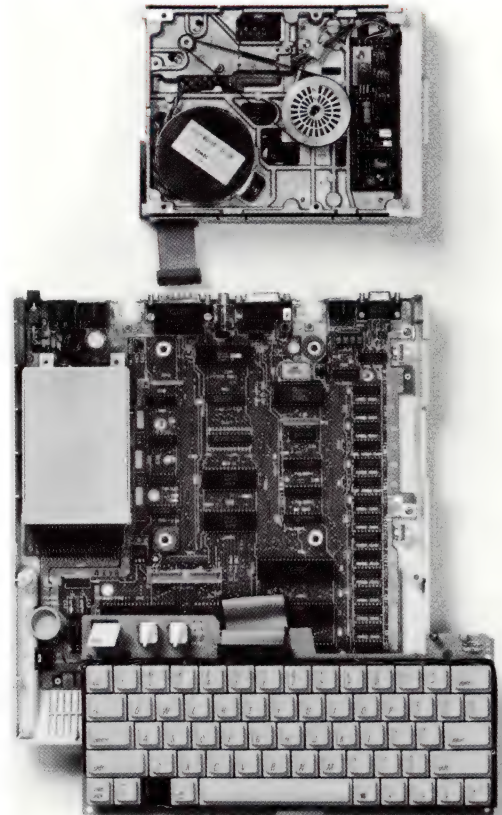
Unused bytes

6.7 Summary of Error Codes

Following is a summary of Protocol Converter call error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the processor status register of the 65C02 microprocessor) is cleared (0), and the accumulator (the A register) contains 0s. If the call was unsuccessful, the C flag is set (1), and the A register contains the error code.

\$00		No error.
\$01	BadCmd	A nonexistent command was issued. Check the command number in the Protocol Converter call.
\$04	BadPCnt	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.
\$06	BusErr	A communications error between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources; make sure the cable is properly shielded.
\$11	BadUnit	Unit number \$00 was used in a call other than STATUS, CONTROL, or INIT.
\$21	BadCtl	The control or status code is not supported by the device.
\$22	BadCtlParm	The control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.
\$27	IOError	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective. Make sure the device is operating correctly.
\$28	NoDrive	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.

\$2B	NoWrite	The medium in the device is write protected.
\$2D	BadBlock	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided vs. double-sided disk, for example).
\$2F	OffLine	Device off-line or no disk in drive. Check the cables and connections; make sure the medium is present in the drive, and that the drive is functioning correctly.
\$30-\$3F	DevSpec	Errors that differ from device to device. See the technical manual for the device in question for details.
\$40-\$4F		Reserved for future expansion.
\$50-\$7F	NonFatal	A device-specific soft error. The operation completed successfully, but some exception condition was detected. See the technical manual for the device in question for details.



Serial port 1 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as an output port for RS-232 devices, such as printers and plotters. It can be changed to a serial communication port (like port 2) either by using the *System Utilities* disk or from a program.

▲Warning

Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

Table 7-1 summarizes the characteristics of this port if used as a printer/plotter port, and is a guide to the other information in this chapter. If you change port 1 to a communication port, refer to the descriptions in Chapter 8, and use 1 instead of 2 for the port number when required.

The serial port back panel connectors are described in Section 11.11.

Table 7-1. Serial Port 1 Characteristics

Port number:	Serial port 1
Commands:	Keyboard command: PR#1
	BASIC command: PR#1
	Monitor command: 1 CONTROL-P (does not work if there is an operating system in RAM)
	All other commands: See Table 7-2
Initial characteristics:	See Section 7.2
Hardware page locations:	See Table 7-3
Monitor firmware routines:	None
I/O firmware entry points:	See Table 7-4
Use of screen holes:	See Table 7-5
Use of other pages:	None

7.1 Using Serial Port 1

Refer to Table 7-4 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

You can access the firmware from BASIC in the usual way—that is, by issuing CONTROL-D (if DOS or ProDOS is in RAM) and PR#1. Subsequent output is directed to the printer (or other device) connected to serial port 1.

To direct Pascal output to the printer, you can use either #6: or PRINTER: .

Your programs can also access the port by changing the value of CSW (see Chapter 3).

Table 7-2 lists the commands you can use with serial port 1, either from a program or from the keyboard, after you issue PR#1.

UniDisk 3.5

Commands followed by an asterisk in Table 7-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).

Each command must be preceded by CONTROL-I (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. You do not have to press **RETURN** after commands that you have entered from the keyboard, or send RETURN from your program if it is sending commands to the port. You can type more than one command on a line, but each must be preceded by the command character.

Table 7-2. Printer Port Commands

Note: The commands themselves are letter commands, not control characters.

Command	Description			
nnn	Sets new line width of nnn (from 1 through 255). This command must be followed by N (see below) or by a carriage return.			
nnB	Sets baud rate to value corresponding to nn:			
	nn	Rate	nn	Rate
	1	50	9	1800
	2	75	10	2400
	3	110 (109.92)	11	3600
	4	135 (134.58)	12	4800
	5	150	13	7200
	6	300	14	9600
	7	600	15	19200
	8	1200		

Table 7-2—continued. Printer Port Commands
Note: The commands themselves are letter commands, not control characters.

Command	Description																											
C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.																											
nD	Sets data format to values corresponding to n: <table><tr><th>n</th><th>Data Bits</th><th>Stop Bits</th></tr><tr><td>0</td><td>8</td><td>1</td></tr><tr><td>1</td><td>7</td><td>1</td></tr><tr><td>2</td><td>6</td><td>1</td></tr><tr><td>3</td><td>5</td><td>1</td></tr><tr><td>4</td><td>8</td><td>2</td></tr><tr><td>5</td><td>7</td><td>2</td></tr><tr><td>6</td><td>6</td><td>2</td></tr><tr><td>7</td><td>5</td><td>2</td></tr></table>	n	Data Bits	Stop Bits	0	8	1	1	7	1	2	6	1	3	5	1	4	8	2	5	7	2	6	6	2	7	5	2
n	Data Bits	Stop Bits																										
0	8	1																										
1	7	1																										
2	6	1																										
3	5	1																										
4	8	2																										
5	7	2																										
6	6	2																										
7	5	2																										
F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.																											
I	Echoes printer output on the screen.																											
K	Disables automatic line feed after carriage return.																											
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.																											
M*	When this command is enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.																											
nnnN	Changes line width to nnn (from 1 through 255; nnn is optional); does not echo printer output on the screen. Note: 0N does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location \$0579, or use <i>System Utilities</i> disk.																											

Table 7-2—continued. Printer Port Commands
Note: The commands themselves are letter commands, not control characters.

Command	Description
nP	Sets parity corresponding to n: <div> <div>n</div> <div>Parity</div> <div>0</div> <div>none</div> <div>1</div> <div>odd</div> <div>2</div> <div>none</div> <div>3</div> <div>even</div> <div>4</div> <div>none</div> <div>5</div> <div>MARK (1)</div> <div>6</div> <div>none</div> <div>7</div> <div>SPACE (0)</div> </div>
R	Resets port 1 (Section 7.2) and exits from serial port 1 firmware.
S	Sends a 233-millisecond BREAK character (used with some printers to synchronize with serial ports).
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF (\$13) character and responds by halting transmission until an XON (\$11) is received. Normally this command is disabled.
Z	Zaps (ignores) further command characters until <div>CONTROL-RESET</div> or PR#1. Does not format output or insert carriage returns into output stream.

* Command (with the exception of L) is available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space), the command is enabled. If you follow the command with D (with no intervening space), command is disabled. The L command is available on the earlier Apple IIc, but cannot be toggled there.

The serial port 1 command character is set as CONTROL-I when the Apple IIc is turned on. You can change it to a different control character by sending the current control character followed immediately by the new control character you want. This is useful if you want to be able to send CONTROL-I to the printer without firmware intervention. For example, to change the command character from CONTROL-I to CONTROL-V, send CONTROL-I CONTROL-V either from the keyboard or a program. (CONTROL-V and CONTROL-W are the recommended substitute control

characters.) To change the command character back again, send CONTROL-V CONTROL-I. Don't slip any spaces between the control characters that you send.

▲Warning

Do not use CONTROL-A, -B, -C, -H, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

The following are examples of valid commands and command sequences. These examples all show commands being entered from the keyboard, but your programs can send the characters just as well. Remember to issue a PR#1 before starting to send commands to serial port 1.

To echo output to the display screen:

CONTROL-I I

To set line width 72, disable line feed, and echo:

CONTROL-I K CONTROL-I 7 2 N

To change control character to CONTROL-V:

CONTROL-I CONTROL-V RETURN

To set up the serial port to allow sending CONTROL-I as part of a character stream:

CONTROL-V (command) RETURN

7.2 Characteristics of Port 1 at Startup

After power-up, the printer firmware sets the following configuration:

- ☐ 9600 baud
- ☐ eight data bits, no parity bits, two stop bits
- ☐ 80-column line width; no echo to display screen
- ☐ firmware supplies line feed after carriage return
- ☐ command character is set to CONTROL-I

These values are stored in the auxiliary memory screen holes (Table 7-5). You can change some of these settings from the keyboard by typing PR#1, the command character, and one of the commands listed in Table 7-2. Section 7.6 describes how port characteristics change as a result of various activities.

7.3 Hardware Page Locations for Port 1

Table 7-3 lists for serial port 1 the addresses and bit assignments of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Section 11.11.

▲Warning

This table is for your information only. To avoid having problems with the system, you should **never** try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.

Table 7-3. Port 1 Hardware Page Locations

ACIA stands for asynChronous communication interfaCe adapter, a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 2.

Location	Description
\$C090-\$C097	Reserved
\$C098	ACIA transmit/receive data register
\$C099	ACIA status register
\$C09A	ACIA command register
\$C09B	ACIA control register
\$C09C-\$C09F	Reserved

7.4 I/O Firmware Support for Port 1

Table 7-4 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Section 3.4.2 describes how to use this protocol.

Table 7-4. Port 1 I/O Firmware Protocol

Address	Value	Description
\$C105	\$38	Pascal ID byte
\$C107	\$18	Pascal ID byte
\$C10B	\$01	Generic signature byte of firmware cards
\$C10C	\$31	Same ID as for Super Serial Card
\$C10D	\$ii	\$C1ii is entry point of initialization routine (PInit).
\$C10E	\$rr	\$C1rr is entry point of read routine (PRead).
\$C10F	\$ww	\$C1ww is entry point of write routine (PWrite).
\$C110	\$ss	\$C1ss is entry point of the status routine (PStatus).
\$C111	nonzero	No optional routines

7.5 Screen Hole Locations for Port 1

The ACIA register bits are defined in Chapter 11.

Table 7-5 lists the screen hole locations that serial port 1 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

Table 7-5. Port 1 Screen Hole Locations

Auxiliary Memory Screen Holes (firmware loads values at power-up):		
Location	Description	
\$0478	\$9E (ACIA control reg: eight data + two stop bits, 9600 baud)	
\$0479	\$0B (ACIA command reg: no parity)	
\$047A	\$40 (flags: no echo, auto LF after CR, serial port)	
	Bit	Interpretation
	7	Echo output on display (0 = no echo)
	6	Generate LF after CR (0 = no LF)
	5-1	Always = 0 (reserved)
	0	1 = communication port; 0 = serial printer port
\$047B	\$50 (printer width: 80 columns)	
	Bit	Interpretation
	7-0	Printer width (0 = do not insert CR)
Main Memory Screen Holes:		
Location	Description	
\$0479	Reserved	
\$04F9	Reserved	
\$0579	Printer width (1-255; 0 = disable formatting)	
\$05F9	Temporary storage location	
\$0679	Bit 7 = 1 while the firmware is parsing a command string	
\$06F9	Current command character (initially CONTROL-I)	
\$0779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return	
\$07F9	Current printer column	

7.6 Changing Port 1 Characteristics

Figure 7-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:


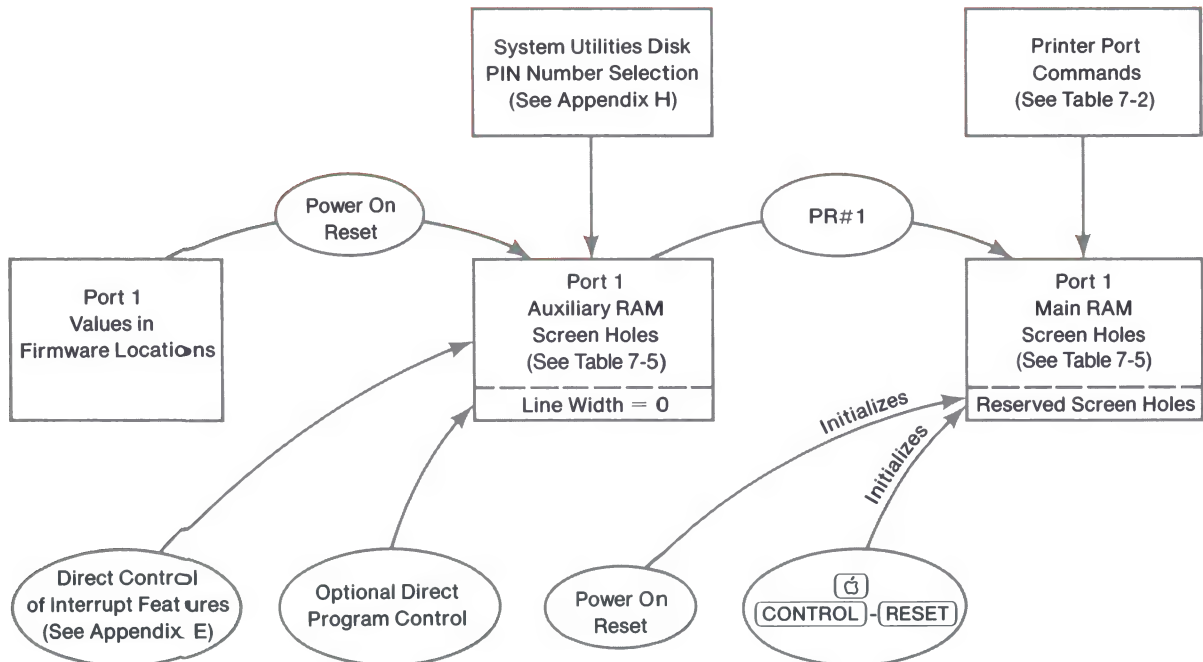
- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Section 7.2 from ROM into the auxiliary memory screen holes listed in Table 7-5.
- If you specify new characteristics using the *System Utilities* disk, the SUD software changes the values in the auxiliary memory screen holes. Your programs can do the same thing.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either -CONTROL-RESET or a simple CONTROL-RESET. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.

Figure 7-1. Diagram of Port 1 Characteristics Storage



- PR#1 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$0579.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 7-2 to change them.

7.6.1 Data Format and Baud Rate

Serial data transfer consists of a string of 1s and 0s sent down a wire at a prearranged rate of transmission, called the baud rate.

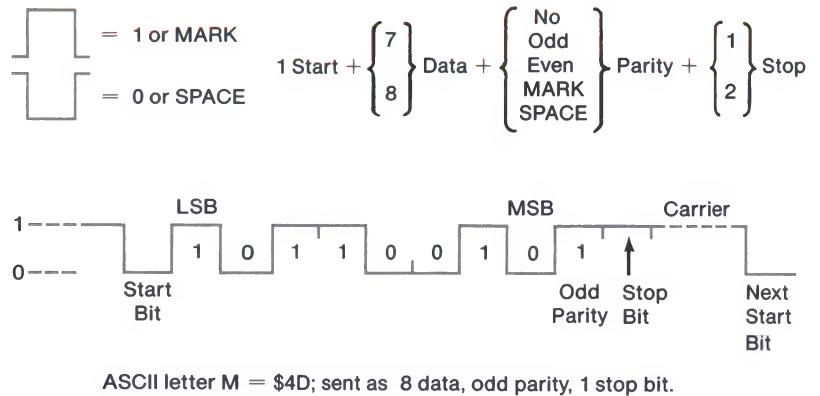
Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the carrier (Figure 7-2). When the value goes to 0, the receiver presumes it is a start bit—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit could possibly last, it is considered a BREAK signal, which some printers use for synchronization.

If the first 0 proves to be a bit, it is interpreted as the start bit. Next come the seven or eight data bits (six is seldom used with computers), low-order bit first. If parity is on, it comes next in the message. Finally, one or two stop bits appear. The stop bits have a value of 1, like the carrier. The next start bit begins transfer of the next character of data.

The parity bit provides a simple check of data validity. Odd parity means the sender counts the number of 1s among the data bits, and sends the appropriate parity bit to make the total number of 1s odd. With even parity, the sender adds the appropriate parity bit to make the total number of 1 bits even. MARK parity is always a 1 bit; SPACE parity is always a 0. The receiver can then check that the parity bit is correct.


If the baud rate is 300, and the data format is one start bit plus seven data bits plus one parity bit plus one stop bit (totaling ten bits transmitted for each byte of data sent), then the actual transfer rate is about 30 characters per second.

Figure 7-2. Data Format



7.6.2 Carriage Return and Line Feed

If you are using a typewriter and you push the carriage all the way to the right (in other words, position the printing mechanism at the left margin), you have performed a carriage return. On the other hand, turning the platen so the paper moves to the next line (or using the index key on an electric typewriter) is called a line feed. Most typewriters perform a line feed automatically after a carriage return, and so the two seem to be one—but they are not.

Carriage return and line feed are separate ASCII codes. Carriage return is sometimes denoted CR; it is ASCII code 13 (\$0D). Line feed, sometimes denoted LF, is ASCII code 10 (\$0A).  on the Apple IIc keyboard generates a LF.

Some printers can supply a line feed automatically after detecting a carriage return; others cannot. If the printer does not supply a line feed after a carriage return and it is not supplied in the data stream, the printer keeps printing over and over on the same line. On the other hand, if both the printer and the Apple IIc firmware supply LF after CR, double line-spacing results.

If the print head keeps moving too far to the right across the page and then prints many characters on top of one another on the right, then the firmware should be instructed to furnish CR after a certain line width has been reached. If the printer prints too short a line before moving to the next line, then probably the firmware is using too small a line width.

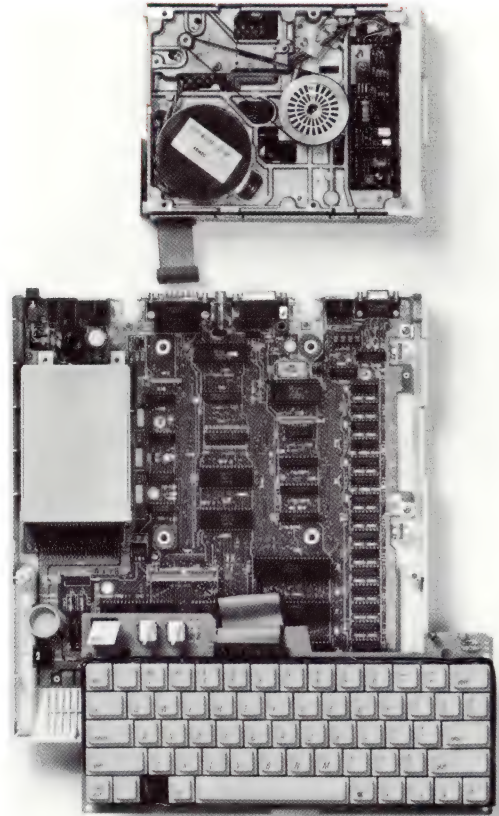
If the printer misses characters at the beginning of each line but otherwise prints correctly, there is probably not enough time for the print mechanism to return to the left margin in response to CR. You must use a lower baud rate with such a printer.

7.6.3 Sending Special Characters

If you want to send special characters (control characters) to the printer without having them intercepted and executed by the Apple IIc firmware, use the Z command (see Table 7-2). If the only special character that causes a problem is the command character (normally CONTROL-I for port 1), you can change just the command character instead of using the zap (Z) command. If you use the zap command, the firmware does no formatting; that is, it does not check line width or insert carriage returns or line feeds. This may be necessary to send graphics to a printer or plotter.

7.6.4 Displaying Output on the Screen

You can display printer output on the screen, but if the printer line width exceeds the 40 or 80 columns you have selected for display, you should turn off video display.



Serial port 2 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as a communication port for modems. You can change it to a serial printer port (like port 1) using the *System Utilities* disk or from a program.

▲Warning

Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

Table 8-1 summarizes the characteristics of this port and is a guide to the other information in this chapter. If you change port 2 to a serial printer port, refer to the descriptions in Chapter 7, and use 2 instead of 1 for the port number when required.

The serial port connectors are described in Section 11.11.

Table 8-1. Serial Port 2 Characteristics

Port number:	Serial port 2
Commands:	Keyboard commands: IN#2 before Table 8-2 commands IN#2 to accept port 2 input PR#1 to echo input to printer PR#2 to echo input back to port 2 BASIC commands: same Monitor command: 2 CONTROL-P (does not work if there is an operating system in RAM) All other commands: See Table 8-2
Initial characteristics:	See Section 8.2
Hardware page locations:	See Table 8-3
Monitor firmware routines:	None
I/O firmware entry points:	See Table 8-4
Use of screen holes:	See Table 8-5
Use of other pages:	In terminal mode, firmware uses auxiliary memory locations \$0800-\$087F to store keyboard input, and \$0880-\$08FF as a serial input buffer.

8.1 Using Serial Port 2

You can access the firmware from BASIC in the usual way—that is, by issuing CONTROL-D (if DOS or ProDOS is in RAM) followed by IN#2 or PR#2. Subsequent input and output are routed through the modem (or other device) connected to serial port 2.

Important!

In terminal mode, the modem port commands listed in Table 8-2 must follow CONTROL-D and IN#2 (*not* PR#2) and the command character (which is usually CONTROL-A).

Refer to Table 8-4 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

To transfer files to the modem under Pascal, specify REMOUT: or #8: . To transfer files from the modem under Pascal, specify REMIN: or #7: .

Table 8-2 lists the commands you can use with serial port 2, either from a program or from the keyboard, after you issue IN#2.

UniDisk 3.5

Commands followed by an asterisk in Table 8-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).

Each command must be preceded by CONTROL-A (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. If you press **RETURN**, you get the current video cursor again. You do not have to press **RETURN** (or send a RETURN character) after commands. You can type more than one command on a line, but each must be preceded by the command character.

Table 8-2. Modem Port Commands

Note: The commands themselves are letter commands, not control characters.

Command	Description																																				
nnn	Sets new line width of nnn (from 1 through 255); this must be followed immediately by N (see below) or by carriage return.																																				
nnB	Sets baud rate to value corresponding to nn: <table><tr><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th></tr><tr><td>1</td><td>50</td><td>6</td><td>300</td><td>11</td><td>3600</td></tr><tr><td>2</td><td>75</td><td>7</td><td>600</td><td>12</td><td>4800</td></tr><tr><td>3</td><td>110 (109.92)</td><td>8</td><td>1200</td><td>13</td><td>7200</td></tr><tr><td>4</td><td>135 (134.58)</td><td>9</td><td>1800</td><td>14</td><td>9600</td></tr><tr><td>5</td><td>150</td><td>10</td><td>2400</td><td>15</td><td>19200</td></tr></table>	nn	Rate	nn	Rate	nn	Rate	1	50	6	300	11	3600	2	75	7	600	12	4800	3	110 (109.92)	8	1200	13	7200	4	135 (134.58)	9	1800	14	9600	5	150	10	2400	15	19200
nn	Rate	nn	Rate	nn	Rate																																
1	50	6	300	11	3600																																
2	75	7	600	12	4800																																
3	110 (109.92)	8	1200	13	7200																																
4	135 (134.58)	9	1800	14	9600																																
5	150	10	2400	15	19200																																

Table 8-2—continued. Modem Port Commands
Note: The commands themselves are letter commands, not control characters.

Command	Description																											
C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.																											
nD	Sets data format to values corresponding to n: <table><tr><th>n</th><th>Data Bits</th><th>Stop Bits</th></tr><tr><td>0</td><td>8</td><td>1</td></tr><tr><td>1</td><td>7</td><td>1</td></tr><tr><td>2</td><td>6</td><td>1</td></tr><tr><td>3</td><td>5</td><td>1</td></tr><tr><td>4</td><td>8</td><td>2</td></tr><tr><td>5</td><td>7</td><td>2</td></tr><tr><td>6</td><td>6</td><td>2</td></tr><tr><td>7</td><td>5</td><td>2</td></tr></table>	n	Data Bits	Stop Bits	0	8	1	1	7	1	2	6	1	3	5	1	4	8	2	5	7	2	6	6	2	7	5	2
n	Data Bits	Stop Bits																										
0	8	1																										
1	7	1																										
2	6	1																										
3	5	1																										
4	8	2																										
5	7	2																										
6	6	2																										
7	5	2																										
F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.																											
I	Echoes output on the screen.																											
K	Disables automatic line feed after carriage return.																											
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.																											
M*	When this command is enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.																											
nnnN	Sets line width to nnn (from 1 through 255); does not echo output on the screen. Note: ON does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location \$057A, or use <i>System Utilities</i> disk.																											

Table 8-2—continued. Modem Port Commands
Note: The commands themselves are letter commands, not control characters.

Command	Description
nP	Sets parity corresponding to n: <div> <div>n</div> <div>Parity</div> <div>0</div> <div>none</div> <div>1</div> <div>odd</div> <div>2</div> <div>none</div> <div>3</div> <div>even</div> <div>4</div> <div>none</div> <div>5</div> <div>MARK (1)</div> <div>6</div> <div>none</div> <div>7</div> <div>SPACE (0)</div> </div>
Q	Quits terminal mode.
R	Resets port 2 (Section 8.2) and exits from serial port 2 firmware.
S	Sends a 233-millisecond BREAK character.
T	Enters terminal mode. Use this command after IN#2 only. Also, if you follow this command by PR#2, the Apple IIc echoes input to output. (If the other device does so too, the first character loops endlessly, locking up the system. Use <code>CONTROL-RESET</code> to get out.)
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF (\$13) character and responds by halting transmission until an XON (\$11) is received. Normally this command is disabled.
Z	Zaps (ignores) further command characters until <code>CONTROL-RESET</code> . Does not format output or insert carriage returns into output stream.
<code>CONTROL-T</code>	This command from a remote device puts the Apple IIc in terminal mode if IN#2 is already in effect. It is the same as <code>CONTROL-A T</code> typed locally.
<code>CONTROL-R</code>	This command from a remote device undoes the terminal mode command. If IN#2 and PR#2 are in effect, the remote keyboard and display become the input and output devices of the local Apple IIc. It is the same as <code>CONTROL-A Q</code> typed locally.

* Command (with the exception of L) available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space) the command is enabled. If you follow the command with D (with no intervening space) the command is disabled. The L command is available on the earlier Apple IIc, but it cannot be toggled there.

When the Apple IIc is turned on, the serial port 2 command character is defined as a CONTROL-A. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send CONTROL-A to the output device without firmware intervention.

For example, to change the command character from CONTROL-A to CONTROL-V, send CONTROL-A CONTROL-V either from the keyboard or a program. (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, send CONTROL-V CONTROL-A.

▲Warning

Do not use CONTROL-B, -C, -H, -I, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

The following are examples of valid commands and command sequences. These examples all show commands being entered from the keyboard, but your programs can send the characters just as well.

To enable echo to the screen:

CONTROL-A I

To send a BREAK character to a remote device:

CONTROL-A B

To change the control character to CONTROL-V (for example, so you can send CONTROL-A as part of a character stream):

CONTROL-A CONTROL-V CONTROL-V (command)

8.2 Characteristics of Port 2 at Startup

After power-up, the firmware sets the following configuration:

- ☐ 300 baud
- ☐ eight data bits, no parity bits, one stop bit
- ☐ firmware does not supply line feed after carriage return
- ☐ firmware does not insert carriage returns into output stream
- ☐ firmware does not echo output to the display screen
- ☐ command character is set to CONTROL-A

These values are stored in the auxiliary memory screen holes (Table 8-5). You can change some of these settings from the keyboard using the command character followed by one of the commands listed in Table 8-2. Section 8.6 describes how port characteristics change as a result of various activities.

If you change any of these values using keyboard commands or commands from a program, subsequent accesses to the port firmware (even by another program) use the new settings instead of the power-up values. This allows you to change the settings once at system startup, and get the desired configuration for subsequent uses. Refer to Section 8.6 for a complete description of these processes.

8.3 Hardware Page Locations for Port 2

Table 8-3 lists for serial port 2 the addresses of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Section 11.11.

▲Warning

This table is for your information only. To avoid having problems with your system, you should **never** try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.

Table 8-3. Port 2 Hardware Page Locations

Location	Description
\$C0A0-\$C0A7	Reserved
\$C0A8	ACIA transmit/receive data register
\$C0A9	ACIA status register
\$C0AA	ACIA command register
\$C0AB	ACIA control register
\$C0AC-\$C0AF	Reserved

ACIA stands for asynchronous communication interface adapter, a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 1.

8.4 I/O Firmware Support for Port 2

Table 8-4 lists the values in the I/O firmware protocol table for serial port 2. This standardized protocol is available for use by any application program. Section 3.4.2 describes how to use this protocol.

Table 8-4. Port 2 I/O Firmware Protocol

Address	Value	Description
\$C205	\$38	Pascal ID byte
\$C207	\$18	Pascal ID byte
\$C20B	\$01	Generic signature byte of firmware cards
\$C20C	\$31	Same ID as for Super Serial Card
\$C20D	\$ii	\$C2ii is entry point of initialization routine (PInit).
\$C20E	\$rr	\$C2rr is entry point of read routine (PRead).
\$C20F	\$ww	\$C2ww is entry point of write routine (PWrite).
\$C210	\$ss	\$C2ss is entry point of the status routine (PStatus).
\$C211	nonzero	No optional routines

8.5 Screen Hole Locations for Port 2

The ACIA register bits are defined in Chapter 11.

Table 8-5 lists the screen hole locations that serial port 2 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

Table 8-5. Port 2 Screen Hole Locations

Auxiliary Memory Screen Holes (firmware loads values at power-up) :		
Location	Description	
\$047C	\$16 (ACIA control reg: eight data + one stop bit, 300 baud)	
\$047D	\$0B (ACIA command reg: no parity)	
\$047E	\$01 (flags: no echo, no auto LF after CR, communication port)	
	Bit	Interpretation
	7	Echo output on display (0 = no echo)
	6	Generate LF after CR (0 = no LF)
	5-1	Always = 0 (reserved)
	0	1 = communication port; 0 = serial printer port
\$047F	\$00 (line length: do not add any CR to output stream)	
	Bit	Interpretation
	7-0	Line length (0 = do not insert CR)

Table 8-5—continued. Port 2 Screen Hole Locations

Main Memory Screen Holes:

Location	Description
\$047A	Reserved
\$04FA	Reserved
\$057A	Line length (1-255; 0 = disable formatting)
\$05FA	Temporary storage location
\$067A	Bit 7 = 1 if and only if the firmware is currently parsing a command string
\$06FA	Current command character (initially CONTROL-I)
\$077A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return
\$07FA	Current column

8.6 Changing Port 2 Characteristics

Figure 8-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:


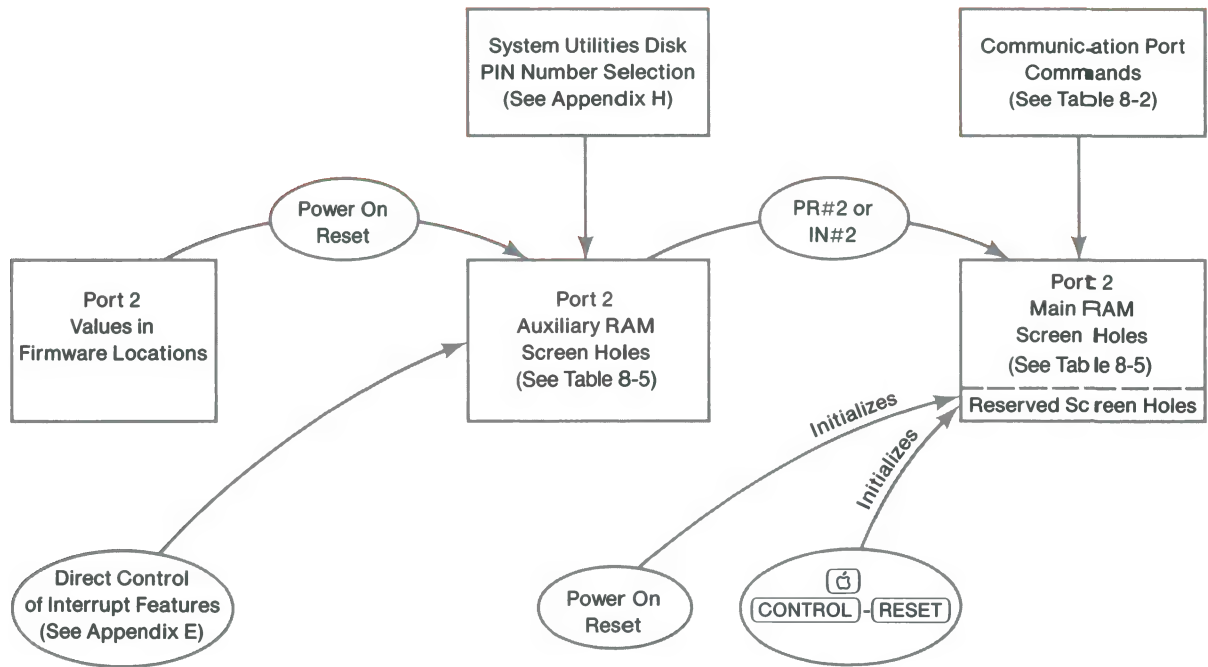
- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 8-2 from ROM into the auxiliary memory screen holes listed in Table 8-5.
- If you specify new characteristics using the *System Utilities* disk, the utility software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either -CONTROL-RESET or a simple CONTROL-RESET. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$057A.

Figure 8-1. Diagram of Port 2 Characteristics Storage



- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 8-2 to change these characteristics.

8.6.1 Data Format and Baud Rate

Section 7.6.1 describes data format and baud rate, and explains how they apply to printers. Refer to that section for definitions of terms.

A noteworthy characteristic of data communication is its strangeness: sometimes the oddest changes make a given communication arrangement work or not work. You must keep this notion firmly in mind when working with serial port 2.

- ☐ the Apple IIc and its firmware, with the baud rate, data format, and other characteristics you have selected
- ☐ the cable from the Apple IIc to the modem
- ☐ the modem
- ☐ possibly an acoustic coupler for a telephone handset
- ☐ the telephone lines, with their switching equipment, boosters, and noise
- ☐ some combination of modem, cable, and remote computer or terminal

Figure 8-2. Devices in a Typical Communication Setup



8.6.2 Carriage Return and Line Feed

If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described as used with printers in Section 7.6.2.

8.6.3 Routing Input and Output

This section discusses the possible ways that serial port 2 can route information. Sometimes the cause of communication problems is that information is not going where you think it is, or it is and you cannot see evidence of the fact. Figures 8-3 through 8-6 show some of the patterns of information flow you can select. This section and the following subsections tell you how to use them.

It is best to read all this material as a unit: questions that arise while you read one description may be answered elsewhere.

The simplest serial port 2 command is IN#2 (Figure 8-3). Port 2 becomes the input device. Data coming into the port gets passed to the input buffer (page \$02 of main memory). Applesoft firmware and system software can see the data and carry out commands in the normal way.

Of course, you can also use just the PR#2 command—for example, if you want to send a listing to the modem.

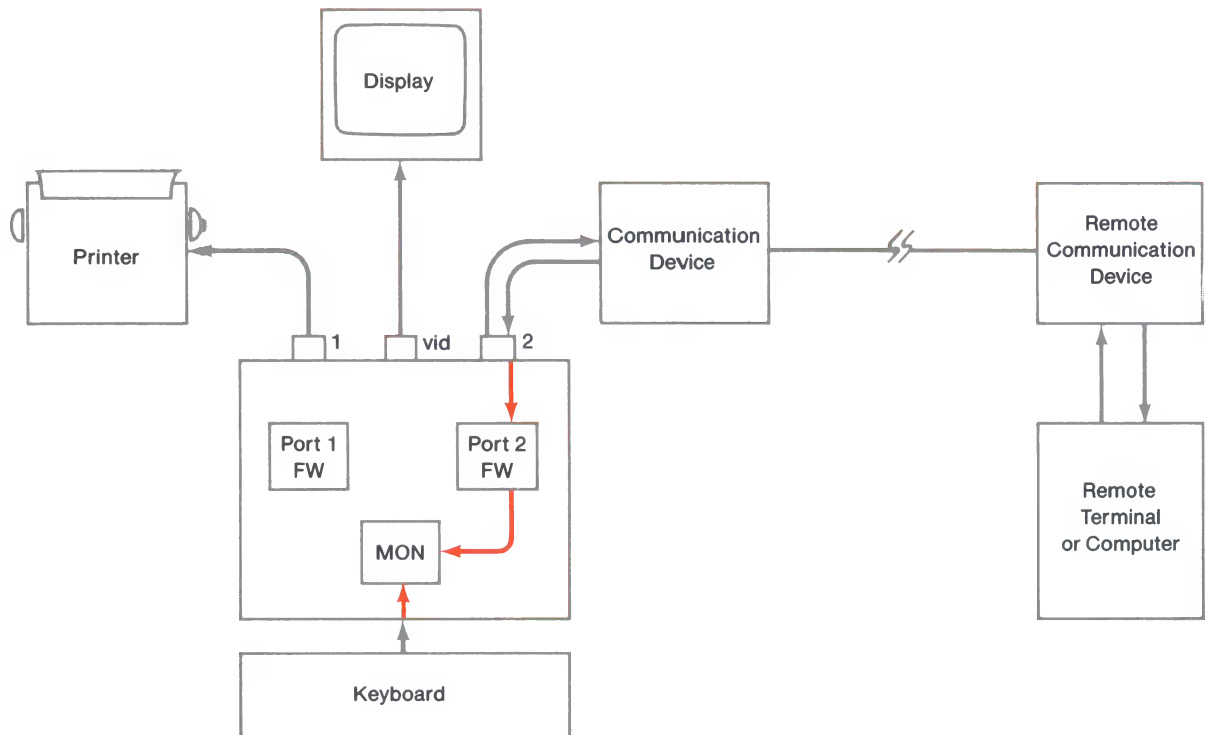
For a further description of what terminal mode does and how to get into and out of it, refer to the last section of this chapter.

To use port 2 for data communication, you ordinarily put it into terminal mode. Following IN#2, pressing **CONTROL-A** gets the attention of the port 2 firmware, which displays a blinking question mark (?) as a prompt. Now type **T** to put the computer in terminal mode. In this mode, the firmware displays a blinking underscore character (_) as a prompt.

In the discussion that follows, *local* refers to your Apple IIc. *Remote* refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

If a remote computer is another Apple IIc or an Apple II series machine with a Super Serial Card in it, then *most* of the commands described here apply to it as well.

Figure 8-3. Effect of IN#2

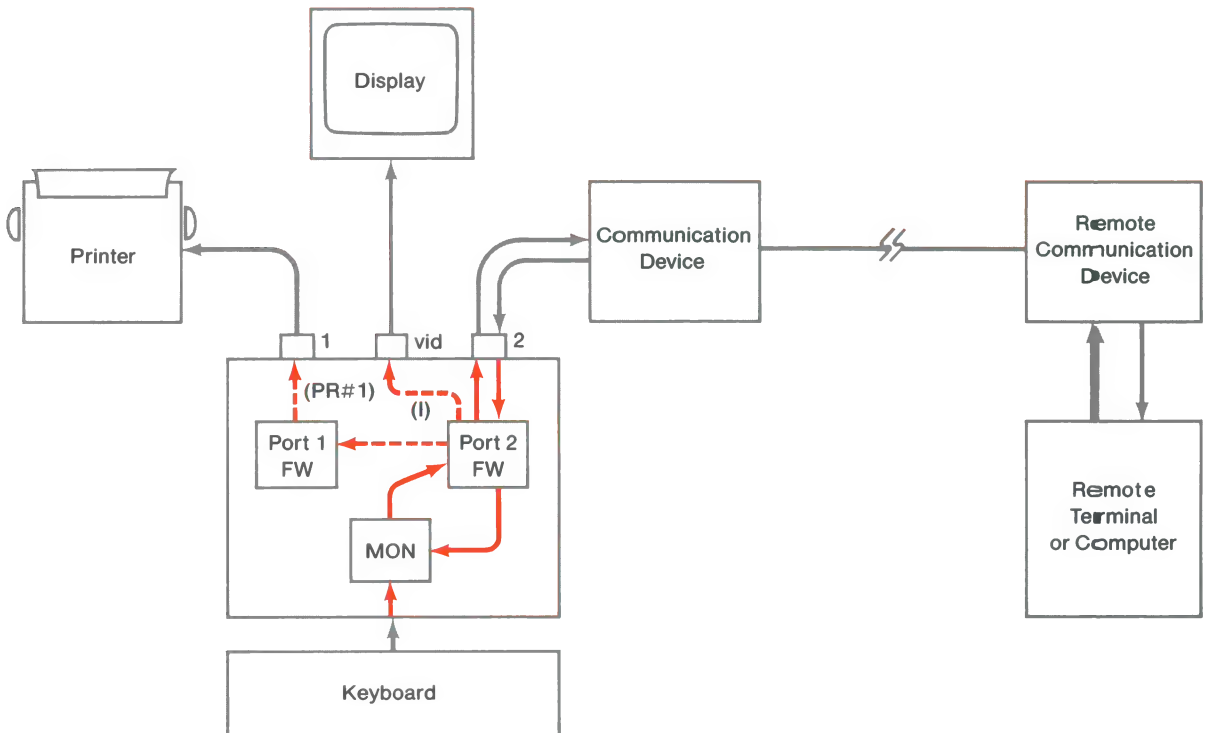


Half-Duplex Operation

In half-duplex operation, information can flow from A to B or from B to A, but in only one direction at a time. In a half-duplex setup, the host does not echo back to the terminal what the terminal sends it. For half-duplex operation, use IN#2 and CONTROL-A T (Figure 8-4) whether the Apple IIc is the host or the terminal.

IN#2 plus CONTROL-A T is the best way to set up the computer for auto-answer operation. The T command allows port 2 firmware to exchange information with the local modem without interference from the local firmware or system software. (The remote device can always cancel the T command with CONTROL-R if necessary, and restore terminal mode with CONTROL-T.) Avoiding PR#2 at this point means that the Apple IIc can operate as a half-duplex terminal, half-duplex host, or full-duplex terminal. (The remote device can also issue CONTROL-A PR#2 if PR#2 is required at the local computer.)

Figure 8-4. Effect of IN#2 and T Command (Half Duplex)



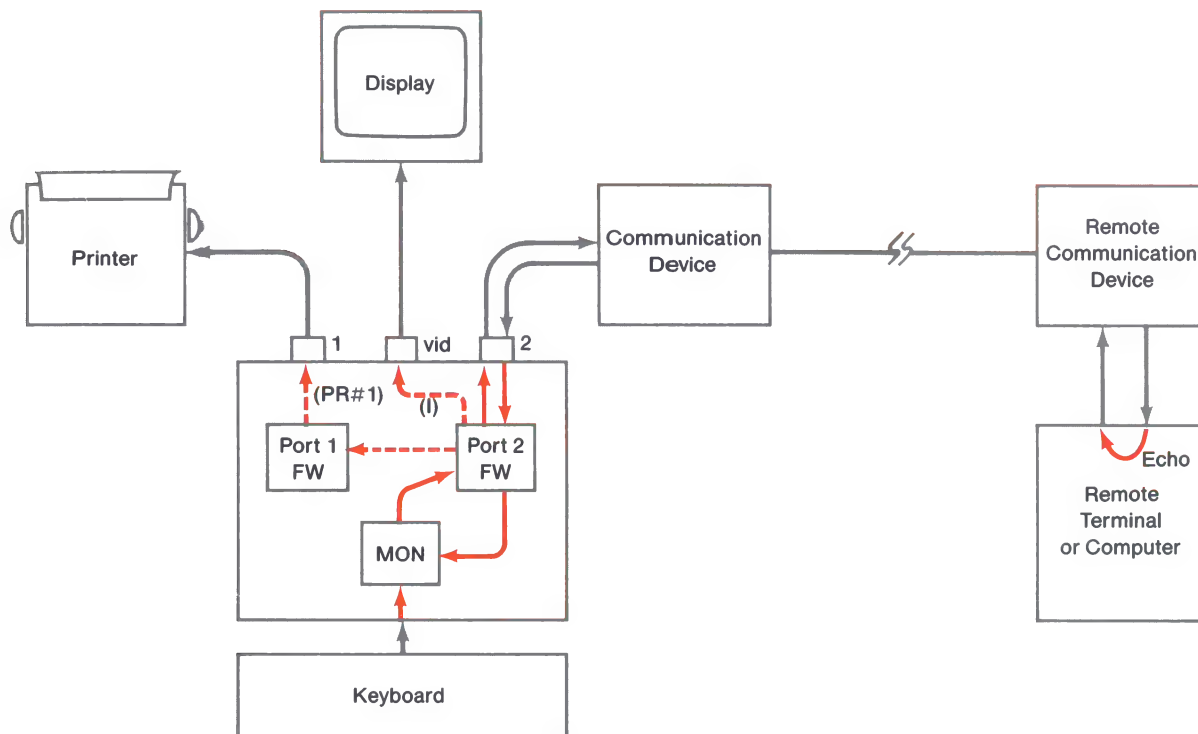
In half-duplex operation, the output hook is available for other uses. For example, you can issue PR#1 to print incoming messages from port 2. Use the CONTROL-A I command to display information on the screen .

Full-Duplex Operation

In full-duplex operation, information can flow from A to B and from B to A simultaneously. Typically, one of the computers (the host computer) echoes its input to output, so the other computer (the terminal) can easily verify that the communication is taking place.

Figure 8-5 shows the flow of information when the Apple IIc is a Full-duplex terminal. (The setup commands, IN#2 and CONTROL-A T, are the same as for half duplex.)

Figure 8-5. Effect of IN#2 and T Command (Full-Duplex Terminal)

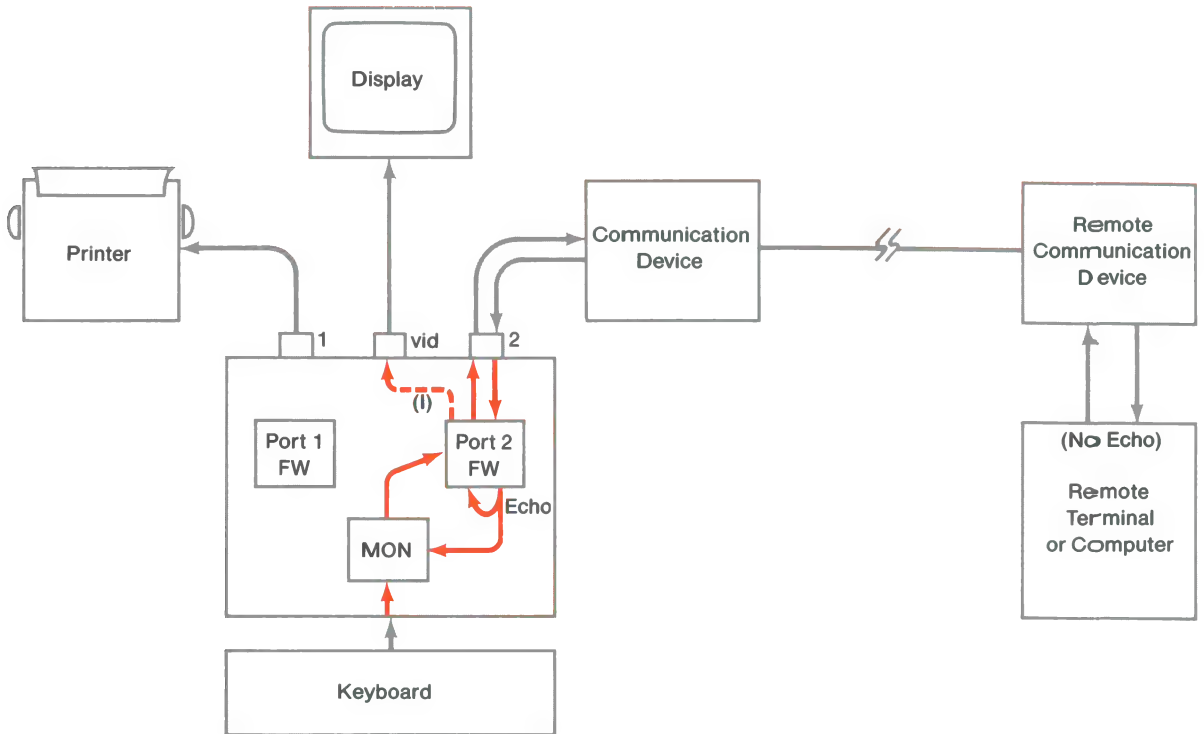


If your Apple IIc is the terminal in full-duplex operation, use the N command to turn off echoing input to the screen. If the Apple IIc does echo input to the screen in this setup, everything you type will appear twice: once from the Apple IIc and once from the host computer.

In this mode of operation, if you echo input to the printer you can get a printed record of both sides of the communication session: the input from the host, and the Apple IIc output as echoed by the host.

Figure 8-6 shows the flow of information when the Apple IIc is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.

Figure 8-6. Effect of IN#2, PR#2, and T Command (Full-Duplex Host)



▲Warning

If the Apple IIc echoes input to output and the other computer does too, then the first subsequent keypress will echo back and forth endlessly and lock up the Apple IIc. This will require a **CONTROL-RESET** to get out.

If you echo input to output when using an information service, the host will end up seeing the echo of what it sent you as though you had typed it.

In this arrangement, the local output hook is not available for using the printer or other device. To display keyboard and port 2 input on the screen, issue **CONTROL-A I**.

Terminal Mode

Terminal mode makes the Apple IIc act like a dumb terminal—one that just sends and receives information, but does not process it. Input and output flow through special serial I/O buffers on page \$08 of auxiliary memory. Applesoft firmware and system software cannot see or interpret the data: only the serial port 2 firmware deals with it.

In most terminal mode setups, the firmware will not display port 2 input unless you use the CONTROL-A I command.

▲Warning

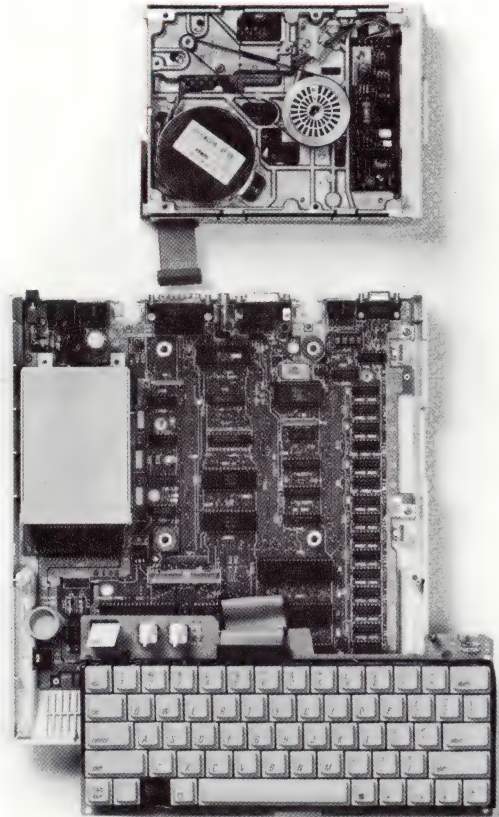
When using terminal mode, \$0800-\$08FF of auxiliary RAM is used for buffering. Any data stored there will be overwritten when terminal mode is enabled.

CONTROL-A T turns on terminal mode, and CONTROL-A Q turns it off.

The remote device can go into terminal mode, and then turn off the local Apple IIc's terminal mode with the CONTROL-R command. If it then issues CONTROL-A PR#2, local output will go to the remote device. The remote keyboard and display then become the input and output devices of the local Apple IIc processor. This is remote mode.

In remote mode, the local Apple IIc does not use the serial I/O buffers (as it does in terminal mode); therefore, local firmware and system software detect and interpret all input and output data. So, for example, if you type **CATALOG** at the remote device keyboard, the local Apple IIc will execute the command and list the disk catalog on the remote device's display. (In terminal mode, the local computer would simply display the word **CATALOG** on its screen.)

The remote device can turn the local Apple IIc's terminal mode back on with CONTROL-T. CONTROL-A T issued at the remote device only turns on the remote device's terminal mode, unless the command character there has already been changed to something else.



This chapter describes the Apple IIc’s mouse port and hand controller (game) input capabilities. The mouse and hand controllers use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands.

A program can tell if a hand controller is connected (Section 9.2) but not if a mouse is connected, unless the computer user moves it.

9.1 Mouse Input

Table 9-1 summarizes the mouse port’s characteristics and guides you to other information in this part of the chapter.

▲Warning

If you want your programs that use the mouse on the Apple IIe and other Apple II series computers to work with the Apple IIc, always use the I/O firmware entry points listed in Tables 9-4 and 9-5, rather than dealing directly with the mouse hardware and RAM locations.

The mouse back panel connector is described in Section 11.12.

Table 9-1. Mouse Input Port Characteristics

Port number:	Mouse input port 4
BASIC commands:	Turn on mouse: PRINT CHR\$(4)“PR#4”:PRINT CHR\$(1) Turn off mouse interrupts: PRINT“PR#4”:PRINT CHR\$(0) Turn on graphics character set: See Section 5.2.2.
Initial characteristics:	After a reset, all mouse inte rrupts are off, and the rising edge of XO and Y0 are selected for interrupts.
Hardware page locations:	See Table 9-2
Monitor firmware routines:	None
I/O firmware entry points:	See Table 9-3 and Table 9-4
Use of screen holes:	See Table 9-5

9.1.1 Mouse Connector Signals

The mouse uses the same DB-9 connector as the hand controllers. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received. Figure 11-37 shows the names of the pin assignments when a mouse is connected.

9.1.2 Mouse Operating Modes

This section tells what the mouse operating modes are for. Later sections of this chapter describe how to set the various mouse operating modes.

Your program should call the `ServeMouse` routine to determine the source of an interrupt as soon as it receives one, in all the interrupt modes except transparent mode.

Transparent Mode

In transparent mode, your program must read screen holes to check for mouse movement. An interrupt routine in the Apple IIc firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task. The findings of the interrupt routine are placed in the screen holes for your program to find. Table 9-5 lists the screen holes with the information that your program should look for.

This is the only mouse mode available to BASIC programs.

Movement Interrupt Mode

On the Apple IIc, a signal called `VBIInt` can interrupt the processor whenever a video vertical blanking signal occurs. This can make it easier for your programs to smoothly move the mouse cursor in response to mouse movements.

In movement interrupt mode, the mouse firmware arms `VBIInt` whenever the mouse is moved at least one count in any direction. When `VBIInt` occurs, program control passes to the vector address contained at locations `$03FE` and `$03FF`; the interrupt handler in your program can then update the cursor smoothly to its next screen position.

Section 5.2.2 contains recommendations for using MouseText characters with a mouse.

Your program's interrupt handler must first call `ServeMouse` (Table 9-3) to see if the mouse caused the interrupt. It should then call `ReadMouse` to get mouse status and its current X-Y position. The routine can also change the mouse mode and position if desired.

The maximum amount of mouse movement that can occur between successive VBL interrupts is limited to the distance someone can move a mouse in one-sixtieth of a second.

Button Interrupt Mode

The Apple IIc mouse-button hardware location does not generate interrupts. However, a program can simulate mouse-button interrupts by polling the button whenever VBLint occurs, and acting on the interrupt whenever the button state has changed. This lets your program provide fast response to the mouse movement without too much program overhead.

Movement/Button Interrupt Mode

The movement/button interrupt mode is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. Your program can effectively process a main task concurrently with cursor and menu updating, as well as menu-selected command processing.

Vertical Blanking Active Modes

The vertical blanking active modes are the same as the four just described except that they allow VBL interrupts to be sent to the user.

9.1.3 Mouse Soft Switches

The soft switches assigned to the mouse interface are shown in Table 9-2. On power-up or reset, the hardware selects the rising edge of X0 and Y0 (mouse movement signals) and masks out all mouse interrupts.

▲Warning

Table 9-2 is included here for your information only. You should use the built-in firmware to access the mouse; doing so is much easier than writing your own mouse interrupt handler, and guarantees compatibility with all other Apple II series computers.

Appendix E explains how the firmware handles interrupts.

Table 9-2. Mouse Soft Switches

Name	Action	Hex	Function
IOUDis	W	\$C07E	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch*
IOUDis	W	\$C07F	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*
RdIOUDis	R7	\$C07E	Read IOUDis switch (1 = off)**
DisXY	R/W	\$C058	Disable (mask) X0 and Y0 movement interrupts***
EnbXY	R/W	\$C059	Enable (allow) X0 and Y0 movement interrupts***
RdXYMsk	R7	\$C040	Read status of X0/Y0 interrupt mask (1 = mask on)
RstXY	R	\$C048	Reset X0/Y0 interrupt flags
X0Edge	R/W	\$C05C	Select rising edge of X0 for interrupt***
X0Edge	R/W	\$C05D	Select falling edge of X0 for interrupt***
RdX0Edge	R7	\$C042	Read status of X0 edge selector (1 = falling)
RstXInt	R	\$C015	Reset mouse X0 interrupt flag
Y0Edge	R/W	\$C05E	Select rising edge of Y0 for interrupt***
Y0Edge	R/W	\$C05F	Select falling edge of Y0 for interrupt***
RdY0Edge	R7	\$C043	Read status of Y0 edge selector (1 = falling)
RstYInt	R	\$C017	Reset mouse Y0 interrupt flag
DisVBl	R/W	\$C05A	Disable (mask) VBL interrupts***
EnVBl	R/W	\$C05B	Enable (allow) VBL interrupts***
RdVBIMsk	R7	\$C041	Read status of VBL interrupt mask (1 = mask on)
RstVBl	R	\$C019	Read and then reset VBLInt flag
PTrig	R/W	\$C070	Reset VBLInt flag; trigger paddle timer


Table 9-2—continued. Mouse Soft Switches

Name	Action	Hex	Function
RdBtn0	R7	\$C061	Read first mouse button status (1 = pressed)†
Rd63	R7	\$C063	Read second mouse button status (0 = pressed)‡
MouX1	R7	\$C066	Read status of X1 (mouse X direction) (1 = high)
MouY1	R7	\$C067	Read status of Y1 (mouse Y direction) (1 = high)

* When IOUDis is on, \$C058–\$C05F do not affect mouse, and \$C05E and \$C05F become DHiRes (Table 5-8).

** Read or write to \$C07x also resets VBIInt and triggers paddle timers.

*** These work only if IOUDis is off.

† This location is also the  key (Table 4-1).

‡ This is also the location of the shift-key mod (Appendix F).

Mouse firmware sets interrupts in response to mode settings under program control. The vertical blanking interrupt (VBIInt) is armed if the mouse button is pushed or moves at least a count of 1 in the X0 or Y0 coordinates. Read \$C070 to reset the VBL interrupt. Because a VBL occurs every sixtieth of a second, that is the maximum time that can elapse before the resulting interrupt can be acknowledged and acted upon.

Software can also select which edge of X0 and Y0 information will cause the XInt or YInt.

When an interrupt has occurred, you can read the direction of the mouse's X1 movement by reading address \$C066 bit 7, and Y1 movement by reading address \$C067 bit 7.

A program can read the status of the soft switches by reading one of the locations \$C040–\$C043 and then testing data bit 7.

Section 11.12 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

If you write your own mouse interrupt handler, it should enable the main bank-switched memory, set up its own IRQ vectors at addresses \$FFFE and \$FFFF, keep track of video modes and the alternate stack, and check for the interrupt source in the same manner as the mouse firmware listed in Appendix I, beginning at address \$C400.

The 32K ROM includes a new feature for programs that need to use mouse interrupts for their own purposes. If your program sets bit 7 of the mouse port mode byte at \$07FC to 1, mouse movement interrupts will be passed to the interrupt handler of your program. VBL interrupts will still be handled by the Apple IIc's firmware. You should use this feature only if the mouse firmware can't keep up with your needs.

9.1.4 I/O Firmware Support for Mouse Input

The Apple IIc supports the mouse with firmware starting at address \$C400. This firmware is necessary because the mouse requires fast, transparent interrupt processing to work effectively.

In assembly language you can use direct firmware support for sophisticated mouse applications. To enable the mouse, first load a mode byte into the accumulator (and \$C4 in X, \$40 in Y), and then do a JSR to the firmware routine called SetMouse (Table 9-3). Valid mode bytes are the following:

\$00	Turns mouse off.
\$01	Sets transparent mode.
\$03	Sets movement interrupt mode.
\$05	Sets button interrupt mode.
\$07	Sets movement or button interrupt mode.
\$08	Turns mouse off, VBLInt active.
\$09	Sets transparent mode, VBLInt active.
\$0B	Sets movement interrupt mode, VBLInt active.
\$0D	Sets button interrupt mode, VBLInt active.
\$0F	Sets movement or button interrupt mode, VBLInt active.

The firmware then initializes the mouse. To read the current position and status of the mouse, first load \$C4 into the X register, load \$40 into the Y register, save processor status, disable interrupts, and then JSR to the firmware routine called ReadMouse (Table 9-3), which stores the information in the port 4 screen holes (Table 9-5).

Table 9-3 lists the mouse port firmware routine offsets. Each address contains the low byte of the entry point of the routine described. The calling setup for all routines (except ServeMouse) is the same: the X register must contain \$C4, and the Y register must contain \$40. When the routine has finished, the A, X, and Y register contents are undefined.

Table 9-3. Mouse Firmware Routines

Location	Offset for	Description
\$C412	SetMouse	Sets the mouse mode to the value in the accumulator. Input: A register contains mode (see \$07FC, Table 9-5). Output: Carry bit = 0 means mode was legal; carry bit = 1 means mode was not legal.
\$C413	ServeMouse	Serves mouse interrupt if needed. Input: X, Y, A registers—doesn't matter. Output: Carry bit = 0 means mouse caused the interrupt; carry bit = 1 means something else caused it. This routine updates \$077C to show which event caused the interrupt (values in Table 9-5).
\$C414	ReadMouse	Updates screen holes to show current mouse X-Y position and button status; clears VBIInt, button and movement interrupt bits in the status byte. Doesn't reenale interrupts until after retrieving position values. Output: Carry bit = 0.
\$C415	ClearMouse	Sets the mouse position to 0, though not necessarily within clamping boundaries; leaves button and interrupt bits in status byte unchanged. Output: Carry bit = 0.
\$C416	PosMouse	Sets the mouse coordinates to new values. Input: X and Y screen holes contain new X and Y positions. Output: Carry bit = 0.
\$C417	ClampMouse	Sets new clamping boundaries (see Table 9-5). Does not affect mouse position or update mouse position screen holes; use ReadMouse to do that. Input: A register = 0 means set new X boundaries; A register = 1 means set new Y boundaries. Output: Carry bit = 0.
\$C418	HomeMouse	Sets the internal mouse position to the upper-left corner of the clamping window. Does not update mouse position screen holes; use ReadMouse to do that.
\$C419	InitMouse	Sets startup internal values; does not update mouse-position screen holes. Output: Carry bit = 0.

Here is a sample sequence of events and calls:

1. Four screen holes contain the mouse's X and Y coordinates, and one contains the status of the last mouse movement (Table 9-5).
2. Call InitMouse.
3. Inhibit interrupts, set up the boundaries you want, then call ClampMouse.
4. Use PosMouse, HomeMouse, or ClearMouse to position the mouse where you want it.
5. Put the mouse mode (see address \$07FC in Table 9-5) that you want to use in the accumulator, then call SetMouse.
6. If you have set one of the interrupt modes, then when an interrupt arrives, call ServeMouse to determine the source of the interrupt.
7. Disable interrupts and call ReadMouse. Retrieve the position values, then reenables interrupts.

Pascal Support

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and later versions use. However, Pascal must use a special attach driver to support the mouse.

Table 9-4. Mouse Port I/O Firmware Protocol

Address	Value	Description
\$C405	\$38	Pascal ID byte
\$C407	\$18	Pascal ID byte
\$C40B	\$01	Generic signature byte of firmware cards
\$C40C	\$20	2 = X-Y pointing device; 0 = identification code
\$C40D		Initialization routine (not implemented; returns error code)
\$C40E		Standard read routine (not implemented; returns error code)
\$C40F		Standard write routine (not implemented; returns error code)
\$C410		Standard status routine (not implemented; returns error code)
\$C411	\$00	Optional routines follow
\$C4FB	\$D6	A mouse identification byte

BASIC and Assembly-Language Support

In BASIC, you must turn the mouse on by printing PR#4 and then CHR\$(1) before you can get input from the mouse. This sets transparent mode. After that, reenables video output with PR#3, and take subsequent input from the mouse by issuing IN#4. The first input statement after that (INPUT X,Y,S) initializes and enables the mouse, and returns a three-element string

+xxxx,+yyyy,+st

representing the x-coordinate, y-coordinate, and status digits.

The coordinates will be integers between 0 and +1023. These are called the clamping boundaries of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when you press a key on the keyboard.

The first digit, s, of the status is 0. The second digit, t, of the status is 1 if the mouse button is still pressed, 2 if it was just pressed, 3 if it was just released, and 4 if it is still released.

To disable the mouse, use these statements:

```
PRINT CHR$(4)"PR#4"  
PRINT CHR$(0)  
PRINT CHR(4)"PR#3"
```

9.1.5 Screen Holes

Table 9-5 lists the screen holes that the mouse firmware uses. Note that the mouse firmware reserves port 5 screen holes for its own use. Also, the auxiliary page counterparts of the port 4 addresses are reserved for startup values.

Important!

Some screen holes are different for the Apple IIe mouse. Refer to Appendix F.

Table 9-5. Mouse Port Screen Hole Locations

Scratch Area:

Location	Description
\$0478	Low byte of clamping minimum
\$04F8	Low byte of clamping maximum
\$0578	High byte of clamping minimum
\$05F8	High byte of clamping maximum

Table 9-5—continued. Mouse Port Screen Hole Locations

Port 4 Screen Holes:		
Location	Description	
\$047C	Low byte of X coordinate	
\$04FC	Low byte of Y coordinate	
\$057C	High byte of X coordinate	
\$05FC	High byte of Y coordinate	
\$067C	Reserved	
\$06FC	Reserved	
\$077C	Status byte	
	Bit	1 Equals
	7	Button down
	6	Button was down on last read and still down
	5	Movement since last read
	4	Reserved
	3	Interrupt from VBIInt
	2	Interrupt from button
	1	Interrupt from movement
	0	Reserved
\$07FC	Mode byte (current mode; mask out bits 4-7 when testing)	
	Bit	1 Equals
	7-4	Reserved
	3	VBIInt active
	2	VBL interrupt on button
	1	VBL interrupt on movement
	0	Mouse active

Port 5 Screen Holes:

Reserved

9.1.6 Using the Mouse as a Hand Controller

You can use the mouse as if it were a set of hand controllers or an X-Y pointing device in port 4. If you turn the mouse on, the Monitor hand controller (game paddle) routines take input from the mouse. This is possible because the mouse and the hand controllers all use the same back panel connector.

You can run a BASIC program that uses the Pdl function to read from the mouse by doing the following, either from the keyboard or from a program:

1. Start up the system with the BASIC program that uses paddles.
2. Type **PR#4** and press **RETURN** to turn on the mouse.
3. Press **CONTROL-A RETURN** to initialize the mouse.
4. Type **PR#0** and press **RETURN** to restore output to the screen.
5. RUN the program.

Play the game using the mouse instead of the paddles.

Important!

Many copy-protected games do not work with a mouse. Also, many games don't use built-in firmware for the paddles.

9.2 Game Input

The Apple IIc supports game paddles, joysticks, and other hand controllers connected to the DB-9 connector on its back panel. Table 9-6 is a summary of game input characteristics.

Table 9-6. Game Input Characteristics



Port number:	None
Commands:	None
Initial characteristics:	Game inputs cannot be disabled.
Hardware page locations:	
\$C061	Switch input 0 and 
\$C062	Switch input 1 and 
\$C063	Mouse button (sense is opposite that of \$C061 to distinguish it from paddle button)
\$C064	Analog input (paddle) 0
\$C065	Analog input (paddle) 1
\$C070	Trigger paddle timer

Table 9-6—continued. Game Input Characteristics

Monitor firmware routines:

Location	Name	Description
\$FB1E	PRead	Reads a paddle position

I/O firmware entry points: None

Use of screen holes: None

9.2.1 The Hand Controller Connector Signals




Several inputs are available to programs or devices from the 9-pin D-type miniature connector on the back of the Apple IIc: two 1-bit inputs, or switches, and two analog inputs.

When you connect a pair of hand controllers to the 9-pin connector, the rotary controllers use two analog inputs, and the pushbuttons use two 1-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.

Complete electrical specifications of these inputs are given in Chapter 11; Table 11-22 shows the connector pin numbers.

Switch Inputs (Sw0 and Sw1)

The two 1-bit inputs can be connected to the output of another electronic device that meets the electrical requirements described in Chapter 11, or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you can read the switch with a PEEK and compare the value with 128. If the value is 128 or greater, the switch is on.

The memory locations for these switches are \$C061, \$C062, and \$C063 (decimal locations 49249 through 49251), as shown in Table 9-6. Switch 0 and switch 1 are permanently connected to  and  on the keyboard; these are the ones connected to the buttons on the hand controllers. Location \$C063 is a second address for the mouse button, so that a program can distinguish it from an  keypress. When the mouse button is pressed, \$C063 (bit 7) goes from 1 to 0, and \$C061 (bit 7) goes from 0 to 1.

Analog Inputs (PdI0 and PdI1)

The two analog inputs are designed for use with 150-K Ω variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit (refer to Section 11.13 for details). The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

A program must first reset the timing circuits before it can read the analog inputs. Accessing memory location \$C070 does this. As soon as you reset the timing circuits, the high bits of the bytes at locations \$C064 through \$C067 are set to 1. If you PEEK at them from BASIC (locations 49252 through 49255), the values will be 128 or greater. Within about three milliseconds, these bits will change back to 0—byte values less than 128—and remain there until you reset the timing circuits again. The exact amount of time each of the bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

9.2.2 Monitor Support for Game Input

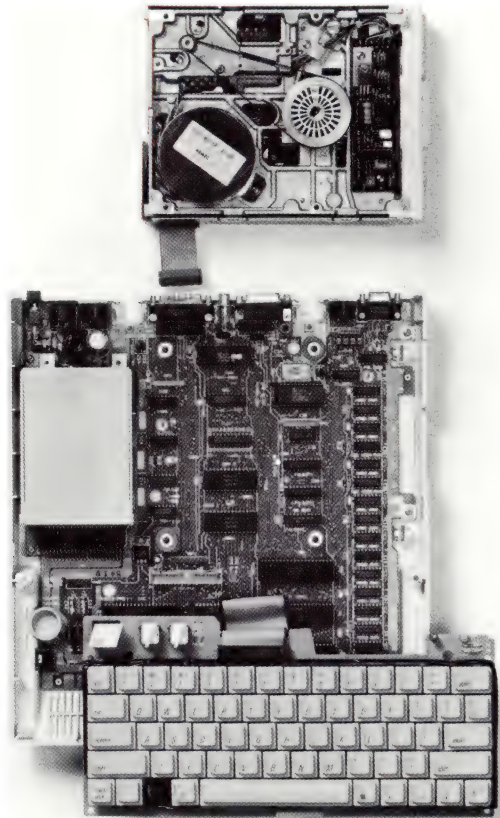
To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0, or you can use the built-in routine PRead. BASIC and other high-level languages also include convenient means of reading the analog inputs—refer to your language manuals. You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least three milliseconds between reading one paddle and reading a different paddle.

The Monitor routine PRead (at address \$FB1E) places in the Y register a number between \$00 and \$FF that represents the position of a hand controller. You pass the number of the hand controller in the X register.

▲Warning

If the hand controller number you furnish in the X register does not equal 0 or 1, strange things may happen.

The paddle and vertical blanking both use \$C070. Disable interrupts before calling PRead if you are reading the paddles and using VBL interrupts.



The System Monitor is a set of subroutines in the Apple IIc firmware that provides a standard interface to the built-in I/O devices described in Chapter 1. Many of the I/O subroutines described in Chapters 3 through 9 are part of the System Monitor.

DOS (but not ProDOS) and the BASIC interpreters (Appendix E) use these subroutines by direct calls to their starting locations. You can call the standard subroutines from your programs in the same fashion. The starting addresses for all of the standard subroutines are listed in Appendix C.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor

- ☐ to look at one or more memory locations
- ☐ to change the contents of any location
- ☐ to write small programs in machine language to be executed **directly** by the Apple IIc
- ☐ to move and compare blocks of memory
- ☐ to invoke other programs from the Monitor.

10.1 Invoking the Monitor

The positive and negative decimal equivalents of Monitor locations are listed in Appendix C. In addition, Appendix H contains conversion tables from one numbering system to another. Appendix E gives further details on how to use Apple IIc firmware from BASIC programs.

The System Monitor starts at memory location \$FF69 (-151). To invoke the Monitor, you make a CALL -151 statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (*), appears on the left side of the display screen, followed by a cursor.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing **CONTROL-RESET**, by pressing **CONTROL-C** and then **RETURN**, or by typing **3D0G**, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location \$03D0.

Important!

If ProDOS (or DOS) is connected via the standard I/O links (Chapter 3), then you can issue commands to it from the Monitor. Under this arrangement, errors will return control to BASIC rather than to the Monitor.

If you want to have **CONTROL-RESET** return you to the Monitor, load the values \$69, \$FF, and \$5A (decimal 105, 255, and 90) into the three locations starting at address \$03F2 (decimal 1010, the reset-vector address and the power-up byte).

10.2 Syntax of Monitor Commands

To give a command to the Monitor, you type a line on the keyboard, then press **[RETURN]**. The Monitor accepts the line using the standard I/O subroutine `GetLn` described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return. It can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading 0s; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are letters (uppercase or lowercase), and some are control characters.

Note: Although the Monitor recognizes and interprets them, control characters typed on an input line do not appear on the screen.

This chapter contains examples of Monitor command use. Some of the data values displayed by your Apple IIc may differ from the values printed in these examples, because they are variables stored in RAM.

10.3 Monitor Memory Commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the last opened location and the next changeable location.

▲Warning

Because locations `$C000` through `$C0FF` contain special hardware circuits, issuing any command that reads or writes on this page can have unpredictable, and perhaps disastrous, results.

10.3.1 Examining Memory Contents

When you type the address of a memory location and press **RETURN**, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000
E000- 4C
*33
0033- AA
*
```

Each time the Monitor displays the value stored at a location, it saves that address as the last opened location and as the next changeable location.

10.3.2 Memory Dump

When you type a period (.) followed by an address, and then press **RETURN**, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger the address after the period is than the last opened location.

```
*20
0020- 00
*.2B
0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07
*300
0300- 99
*.315
0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03
*.32A
```



```

0316- 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

When the Monitor performs a memory dump, it starts at the address immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of 8—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a memory range.

```
*300.32F
```

```

0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
*30.40
```

```

0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
```

```
*E015.E025
```

```

E015- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing **RETURN** by itself makes the Monitor display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-8 boundary. The Monitor saves the address of the last location displayed as both the last opened location and the next changeable location.

```
*5
0005- 00
*RETURN
00 00
*RETURN
0008- 00 00 00 00 00 00 00 00
*32
0032- FF
*RETURN
AA 00 C2 05 C2
*RETURN
0038- 1B FD D0 03 3C 00 3F 00
*
```

10.3.3 Changing Memory Contents

Section 10.3.2 showed you how to *display* values stored in the Apple IIc's memory; this section shows you how to *change* these values. You can change any location in RAM; you can change the characteristics and treatment of an output device by changing the contents of locations assigned to it; and you can change a soft switch setting by referencing its set and reset addresses.

▲Warning

Use these commands carefully. If you change the zero-page locations used by the interpreter or operating system (Appendix B), you may lose programs or data stored in memory.

Changing One Byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon followed by a value.

```
*0
```

```
0000- 4C
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
0000- 5F
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the example, you type the address again to verify the change.

```
*302:42
*302
0302- 42
*
```

When you change the contents of a location, the value that was contained in that location is replaced by the new value, which will remain until you or some program replaces it with another value.

ASCII Input Mode: The Monitor has a tool to make entering values a little easier: ASCII input mode. ASCII input mode lets you enter ASCII characters as well as their hexadecimal ASCII equivalents. This means that 'A is the same as C1 and 'B is the same as C2 to the Monitor. The ASCII value for *any* character following an apostrophe is used by the Monitor. For example, to enter the string "Good morning!" at \$0300 in memory, type

```
*300:'G 'o 'o 'd ' 'm 'o 'r 'n 'i 'n 'g '!
```

Note that each character to be placed in memory is delimited by a leading and a trailing space. The only exception to this rule is that the last character in the line is followed by a RETURN character instead of a space.

Changing Consecutive Locations

You don't have to type a separate command with an address, a colon, a value, and press **RETURN** for each location you want to change. You can change the values of up to 85 consecutive locations at a time—or even more, if you omit leading 0s from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with **RETURN**. The Monitor stores the consecutive values in consecutive

locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations, without typing an address on the next input line, by typing another colon and more values. In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 03
*300
0300- 69
*RETURN
01 20 ED FD 4C 00 03
*10:0 1 2 3
*:4 5 6 7
*10.17
0010- 00 01 02 03 04 05 06 07
*
```

10.3.4 Moving Data in Memory

You can copy a contiguous block of data from one area in the Apple IIc's memory to another in RAM by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory—the source locations—and where you want the copy to go—the destination locations.

The format of the complete MOVE command looks like this:

```
[destination] < [start] . [end] M
```

The destination is the address where you want the first of the moved data to go. The less-than symbol (<) separates the destination address from the starting and ending addresses of the block of data to be moved. The period between two addresses is the Monitor's standard notation for specifying address ranges. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

When you type the actual command, replace the words in braces with hexadecimal addresses, and omit the braces and spaces. Here are some examples of memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory. The actual MOVE commands end with **[M]**.

***0.F**

0000- 5F 00 05 07 00 00 00 00

0008- 00 00 00 00 00 00 00 00

***300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03**

***300.30C**

0300- A9 8D 20 ED FD A9 45 20

0308- DA FD 4C 00 03

***0<300.30C [M]**

***0.C**

0000- A9 8D 20 ED FD A9 45 20

0008- DA FD 4C 00 03

***310<8.A [M]**

***310.312**

0310- DA FD 4C

***2<7.9 [M]**

***0.C**

0000- A9 8D 20 DA FD A9 45 20

0008- DA FD 4C 00 03

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location.

If the destination address of the MOVE command is inside the source range of addresses, then strange things happen: the locations between the beginning of the source range and the destination address are treated as a subrange and the values in this subrange are replicated throughout the source range. Try it.

See Section 10.6 for an interesting application of this feature.

10.3.5 Comparing Data in Memory

You can use the VERIFY command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the VERIFY command can be used immediately after a MOVE to make sure that the move was successful.

The VERIFY command, like the MOVE command, needs a range and a destination. The syntax of the VERIFY command is

|destination| < |start| . |end| ☐

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to \$0D, copy them to locations starting at \$0300 with the MOVE command, and then compare them using the VERIFY command. When you use the VERIFY command after you change the value at location 6 to \$E4, it detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5
```

```
*300<0.D ☐
```

```
*300<0.D ☐
```

```
*6:E4
```

```
*300<0.D ☐
```

```
0006-E4 (EE)
```

```
*
```

If the VERIFY command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. The VERIFY command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the MOVE command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges are compared. Like the MOVE command, the VERIFY command does unusual things if the destination address is within the source range; see Section 10.6.

10.4 Monitor Register Commands

Even though the actual contents of the 65C02's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (BRK). You can also store new register values that will be used when you execute a program from the Monitor using the GO command, described below.

10.4.1 Changing Registers

When you call the Monitor, it stores the contents of the 65C02 registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S (stack pointer), starting at location \$45. When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

10.4.2 Examining Registers

Pressing **CONTROL-E** and then **RETURN** invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
* CONTROL-E
```

```
M=00 A=0A X=FF Y=D8 P=B0 S=F8
```

```
*:B0 02
```

```
* CONTROL-E
```

```
M=00 A=B0 X=02 Y=D8 P=B0 S=F8
```

```
*
```

In the EXAMINE command's display, M shows the current memory state register contents. The memory state register is location \$44, and its interpretation is given in Section E.4 of Appendix E.

10.5 Miscellaneous Monitor Commands

Monitor commands discussed in this section let you do the following:

- change the video display format from normal to inverse and back
- assign input and output to various devices
- leave the Monitor and return to the currently-loaded operating system (DOS 3.3 or ProDOS) or BASIC

10.5.1 Display Inverse and Normal

The COut subroutine is described in Chapter 3.

You can control the setting of the inverse-normal mask location used by the COut subroutine from the Monitor so that all the Monitor's output will be in inverse format. The INVERSE command (I) sets the mask so that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command (N).

*0.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*I

*0.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*N

*0.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*

10.5.2 Back to BASIC

See Appendix D.

If you are using one of the Apple disk operating systems (ProDOS or DOS), press **CONTROL-RESET** or type **3D0G** to return to the language you were using, with your program and variables intact.

Important!

If you type the latter command, make sure that the third character you type is a *zero*, not a letter *O*. The letter *G* is the Monitor's GO command, described in Section 10.7.

If there is no operating system in RAM, use the BASIC command CONTROL-B to leave the Monitor and enter the BASIC interpreter that was active when you entered the Monitor. (Normally this is Applesoft BASIC.) Any program or variables that you had previously in BASIC will be lost. If you want to reenter BASIC with your previous program and variables intact, use the CONTINUE BASIC command (CONTROL-C).

10.5.3 Redirecting Input and Output

The CONTROL-P command diverts all output normally destined for the screen (port 0) to a device attached to one of the other ports, from 1 to 7. The format of the command is

{port number} CONTROL-P

A CONTROL-P command to port number 0 switches the stream of output characters back to the Apple IIc's video display. However, use ESC CONTROL-Q if the enhanced video firmware is active (solid-block cursor).

CONTROL-K controls the input stream in much the same way as CONTROL-P controls the output stream. The format for the command is

{port number} CONTROL-K

Pressing **[0]** **[CONTROL]** **[K]** directs the Monitor to accept input from the Apple IIc's built-in keyboard.

The CONTROL-P and CONTROL-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

10.5.4 Hexadecimal Arithmetic

You can use the Monitor as a one-byte hexadecimal addition and subtraction calculator. Just type a line in one of these formats followed by **[RETURN]**:

{value} + {value} **[RETURN]**
{value} - {value} **[RETURN]**

The Apple IIc performs the arithmetic and displays the result, as shown in these examples.

```
*20 + 13
=33
*4A - C
=3E
*
```

Chapter 3 lists the Apple IIc port numbers available.

For more information on the way those commands work, refer to Section 3.1.

10.6 Advanced Operations

This section describes some ways of using the Monitor commands to speed up your work.

10.6.1 Multiple-Command Lines

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces, and the total number of characters in the line is less than 254. Adjacent single-letter commands such as L, S, I, and N need not be separated by spaces.

You can freely intermix all of the commands except the STORE (:) command. Since the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a STORE must be followed by a letter command before another address is encountered. You can use the NORMAL command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 [N] 300.302
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then stops with a beep and ignores the remainder of the input line.

10.6.2 Filling Memory

The MOVE command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range:

```
*300:11 22 33
*
```

Remember the number of values in the pattern: in this case, it is three. Use the number to compute addresses for the MOVE command, like this:

|start+number| < |start| . |end-number| **[M]**

This MOVE command first replicates the pattern at the locations immediately following the original pattern, then replicates that pattern following itself, and so on until it fills the entire range.

***303<300.32D [M]**

***300.32F**

0300- 11 22 33 11 22 33 11 22

0308- 33 11 22 33 11 22 33 11

0310- 22 33 11 22 33 11 22 33

0318- 11 22 33 11 22 33 11 22

0320- 33 11 22 33 11 22 33 11

0328- 22 33 11 22 33 11 22 33

You can use the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, to see the VERIFY command detect the discrepancy, you first fill the memory range from 0300 to 0320 with 0s and verify it, then change one location and verify again:

***300:0**

***301<300.31F [M]**

***301<300.31F [V]**

***304:02**

***301<300.31F [V]**

0303-00 (02)

0304-02 (00)

10.6.3 Repeating Commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, n=0. The value for n must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location \$0200. Each time the Monitor executes a command, it stores the value of the index at location \$34; when that command is finished, the Monitor reloads the index register with the value at location \$34. By making the last command change the value at location \$34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press **CONTROL-RESET**; that is how this example ends.

```
*N 300 302 34:0 N
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300
*
```

10.6.4 Creating Your Own Commands

The **USER** command (**CONTROL-Y**) forces the Monitor to jump to memory location \$03F8. You can put a **JMP** instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the **CONTROL-Y**. The program starts at location \$0300; the command line that starts with \$03F8 stores a jump to \$0300 at location \$03F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF
*3F8:4C 00 03
*CONTROL-Y THIS IS A TEST
THIS IS A TEST
*
```

10.7 Machine-Language Programs

The main reason to program in machine language is to get more speed and sometimes to also save memory space. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

Note: If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it, and study one of the books on 65C02 programming listed in the bibliography.

You can get a hexadecimal dump of your program or move it around in memory using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

10.7.1 Running a Program

The Monitor command to start execution of your machine-language program is the GO command. When you type an address and press **[G]**, the Apple IIc starts executing machine-language instructions starting at the specified location. If you just press **[G]**, execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters A through Z: you store it starting at location \$0300, examine it to be sure you typed it correctly, then type **300G** to start it running.

```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
*300.30C
```

```

0300- A9 C1 20 ED FD 18 69 01
0308- C9 DB D0 F6 60
*300 G
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

10.7.2 Disassembled Programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called assemblers.

Programs like the Monitor's LIST command are called disassemblers. This command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or mnemonic, and a formatted hexadecimal operand. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format

```
{location} L
```

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenful (20 lines) of instructions, as shown in the following example:

```

*300 L

0300-      A9 C1          LDA      #$C1
0302-      20 ED FD      JSR      $FDED
0305-      18           CLC
0306-      69 01        ADC      #$01
0308-      C9 DB        CMP      #$DB
030A-      D0 F6        BNE      $0302
030C-      60           RTS
030D-      00           BRK
030E-      00           BRK
030F-      00           BRK
0310-      00           BRK
0311-      00           BRK
```


0312-	00	BRK
0313-	00	BRK
0314-	00	BRK
0315-	00	BRK
0316-	00	BRK
0317-	00	BRK
0318-	00	BRK
0319-	00	BRK
*		



The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has 0s in it; other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it displays another screenful of instructions, starting where the previous display left off.

10.8 The STEP and TRACE Commands

UniDisk 3.5

Section 10.8 applies only to the version of the Apple IIc that supports UniDisk 3.5.

STEP and TRACE are Monitor facilities for debugging assembly-language programs. The STEP command decodes, displays, and executes one instruction at a time, and the TRACE command steps continuously through a program, stopping when a BRK instruction is executed or  is pressed. You can press  to slow down the trace to one step per second.

Each STEP command causes the Monitor to execute the instruction in memory pointed to by the program counter. The instruction is displayed in its disassembled form, then executed. The contents of the 65C02's internal registers are displayed after the instruction is executed. After execution, the program counter is incremented to point to the next instruction in the program.

Here is an example of the STEP command, using the following program:

```
$0300:    LDX #02
$0302:    LDA $00,X
$0304:    STA $10,X
$0306:    DEX
$0307:    STA $C030
$030A:    BPL $0302
$030C:    BRK
```

To step through this program, first call the Monitor by typing **CALL - 151** and pressing **RETURN**, and then from the Monitor, type **300S** (to start the STEP routine at address \$0300). Type **S** to advance each additional step through the program. The Monitor keeps the program counter and the last opened address separate from one another, so you can examine or change the contents of memory while you are stepping through your program.

Here's what happens when you step through the program above, examining the contents of location \$0012 after the third step. Note that in this example, what you type appears just after the ***** prompt, and the information on the next two lines—that begin without the ***** prompt—is what the computer displays on the screen in response.

```
*300S

0300-    A2 02    LDX #02
M=CA A=0A X=02 Y=D8 P=30 S=F8
*S

0302-    B5 00    LDA $00,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*S

0304-    95 10    STA $10,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
* 12

0012-    0C
*S

0306-    CA      DEX
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S



0307-    8D 30 C0 STA $C030
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
```

```

030A-    10 F6      BPL $0302
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S


0302-    B5 00      LDA $00,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*S

0304-    95 10      STA $10,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*
```

The TRACE command is a continuous version of the STEP command; it stops stepping through the program only when you press , or when it encounters a BRK instruction in the program. Press  to slow the trace to one step per second.

Important!

Keep the following cautions in mind when using the STEP and TRACE Monitor commands:

- ☐ If the program ends with an RTS instruction, the TRACE routine will continue to run indefinitely until stopped with .
- ☐ You can't step or trace through routines that use the same zero page locations as the Monitor.

10.9 The Mini-Assembler

UniDisk 3.5

Section 10.9 applies only to the version of the Apple IIc that supports UniDisk 3.5.

Without an assembler, you have to write your machine-language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the Monitor commands described in Sections 10.1 through 10.6 in this chapter.

The Mini-Assembler lets you enter machine-language programs directly from the keyboard of your Apple. ASCII characters can be entered in Mini-Assembler programs, exactly as you enter them in the Monitor.

Note that the Mini-Assembler doesn't accept labels; you must use actual values and addresses.

10.9.1 Starting the Mini-Assembler

To start the Mini-Assembler, first invoke the Monitor by typing `CALL - 151` `RETURN`, and then from the Monitor, type `!` followed by `RETURN`. The Monitor prompt character then changes from `*` to `!`.

When you finish using the Mini-Assembler, press `RETURN` from a blank line to return to the Monitor.

To enter code into memory, type the address, a colon, and the instruction. For example, after entering the Mini-Assembler, you could type:

```
!300:STA C030
```

You can enter a series of instructions by typing a space, followed by the instruction, followed by `RETURN`:

```
!300:STA C030
! LDA #A0
! INX
```

Each succeeding instruction is placed in the next available memory location. As you type in instructions, each is replaced by the starting address of the instruction, the hexadecimal value(s) of the instruction, followed by mnemonics describing the instruction. For example, the sequence of instructions given above would produce the following on your screen:

```
0300-    8D 30 C0      STA $C030
0303-    A9 A0       LDA #$A0
0305-    E8          INX
```

When you're ready to execute your program, press `RETURN` to leave the Mini-Assembler and return to the Monitor. Monitor commands can't be executed directly from the Mini-Assembler.

10.9.2 Using the Mini-Assembler

The Mini-Assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-Assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press **RETURN**. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the Mini-Assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press **RETURN**. The Mini-Assembler assembles that line and waits for another.

If the line you type has an error in it, the Mini-Assembler beeps loudly and displays a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The Mini-Assembler also rejects the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler flags this as an error.

Dollar Signs: In this manual, dollar signs (\$) in addresses signify that the addresses are in hexadecimal notation. The dollar signs are ignored by the Mini-Assembler and can be omitted in programs.

!300:LDX #02

0300- A2 02 LDX #\$02
! LDA \$00,X

0302- B5 00 LDA \$00,X
! STA \$10,X

0304 95 10 STA \$10,X
! DEX

0306- CA DEX
! STA \$C030

0307- 8D 30 C0 STA \$C030
! BPL \$0302

030A- 10 F6 BPL \$0302
! BRK

030C- 00 BRK
!

To leave the Mini-Assembler and reenter the Monitor, press **RETURN** at a blank line.

Your assembly-language program is now stored in memory. You can display it with the LIST command:

```
*300L
0300- A2 02      LDX  #$02
0302- B5 00      LDA  $00,X
0304- 95 10      STA  $10,X
0306- CA        DEX
0307- 8D 30 C0   STA  $C030
030A- 10 F6      BPL  $0302
030C- 00        BRK
030D- 00        BRK
030E- 00        BRK
030F- 00        BRK
0310- 00        BRK
0311- 00        BRK
0312- 00        BRK
0313- 00        BRK
0314- 00        BRK
0316- 00        BRK
0316- 00        BRK
0317- 00        BRK
0318- 00        BRK
0319- 00        BRK
*
```

10.9.3 Mini-Assembler Instruction Formats

The Apple IIc Mini-Assembler recognizes 66 mnemonics and 15 addressing formats. The mnemonics are standard, as used in the *Synertek Programming Manual* (Apple part number A2L0003), but the addressing formats are somewhat different, as shown in Table 10-1.

An address consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-Assembler adds leading 0s; if the address has more than four digits, then it uses only the last four.

Table 10-1. Mini-Assembler Address Formats

Addressing Mode	Format
Accumulator	*
Implied	*
Immediate	#{value}
Absolute	\${address}
Zero page	\${address}
Indexed zero page	\${address},X \${address},Y
Indexed absolute	\${address},X \${address},Y
Relative	\${address}
Indexed indirect	(\${address}),X
Indirect indexed	(\${address}),Y
Absolute indirect	(\${address})

* These instructions have no operands.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the Mini-Assembler that can be used in both absolute and zero-page mode, the Mini-Assembler assembles that instruction in absolute mode if the operand for that instruction is greater than \$FF, and it assembles it in zero-page mode if the operand is less than \$0100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-Assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the Mini-Assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the Mini-Assembler will not accept the line.

10.10 Summary of Monitor Commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

10.10.1 Examining Memory

{adrs} **RETURN**

Displays the value contained in one location.

{adrs1}|.{adrs2} **RETURN**

Displays the values contained in all locations between {adrs1} and {adrs2}.

RETURN

Displays the values in up to eight locations following the last opened location.

{adrs} **L**

Lists disassembled code starting at {adrs} and continuing until the screen is full.

10.10.2 Changing the Contents of Memory

{adrs}|:{val} {val}...

STORE command. Stores the values in consecutive memory locations starting at {adrs}.

:{val}|{val}...

Stores values in memory starting at the next changeable location.

10.10.3 Moving and Comparing

{dest|<|start|.end|**M**}

MOVE command. Copies the values in the range {start|.end| into the range beginning at {dest|.

{dest|<|start|.end|**V**}

VERIFY command. Compares the values in the range {start|.end| to those in the range beginning at {dest|.

10.10.4 The Register Command

CONTROL-**E**

EXAMINE command. Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.

10.10.5 Miscellaneous Monitor Commands

I

INVERSE command. Sets inverse display mode.

N

NORMAL command. Sets normal display mode.

CONTROL-**B**

BASIC command. Enters the language currently active (normally Applesoft).

CONTROL-**C**

CONTINUE BASIC command. Returns to the language currently active (normally Applesoft).

{val|+|val|}

Adds the two values and prints the hexadecimal result.

{val|-|val|}

Subtracts the second value from the first and prints the result.

{port| **CONTROL** **P**

Redirects output to the device connected to port number {port|. If {port|=0, sends output to the video display. Use only when the enhanced video firmware is not active (checkerboard cursor).

ESC **CONTROL** **Q**

Redirects output to video display when enhanced video firmware is active (solid block cursor).

{port| **CONTROL** **K**

Takes input from the device connected to port number {port|. If {port|=0, accepts input from the keyboard.

CONTROL **Y**

USER command. Jumps to the machine-language subroutine at location \$03F8.

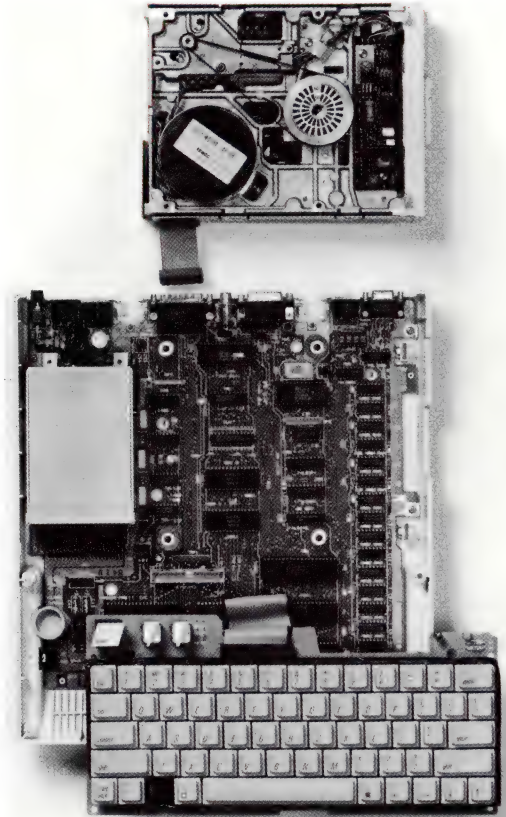
10.10.6 Running and Listing Programs

{adrs| **G**

Transfers control to the machine language program beginning at {adrs|.

{adrs| **L**

Disassembles and displays 20 instructions starting at {adrs|. Subsequent **L**'s display 20 more instructions each.



Most of this manual describes functions—what the Apple IIc does. This chapter, on the other hand, describes objects: the pieces of hardware the Apple IIc uses to carry out its functions. If you are designing a device to connect to the Apple IIc back panel, or if you just want to know more about how the Apple IIc is built, you should study this chapter.

11.1 Environmental Specifications

The Apple IIc is quite sturdy when used in the way it was intended—as a transportable computer, made for use in an indoor environment. You can carry it by its handle from room to room, but for longer trips you should use its carrying case or some other protective container (such as an attache case).

Table 11-1 defines the conditions under which the Apple IIc is designed to function properly.

Table 11-1. Environmental Specifications

Operating temperature:	10° to 40° C (50° to 104° F)
Relative humidity:	20% to 95%
Line voltage:	105 to 129 VAC (normal USA voltage range)

You should treat the Apple IIc with the same kind of care as any other electrical appliance; protect it from physical abuse, and be careful not to bump it against furniture when you move it around. Put it in an attache case or other protective covering if you carry it outside. You should also protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, particularly those with dissolved contaminants, such as soups, fruit juices, and carbonated soft drinks.

In normal operation (with the handle locked in its down position), enough air flows through the openings in the case to keep the insides from getting too hot. If you do overheat your Apple IIc—for example, by blocking the upper or lower ventilation openings—the first symptom will be erratic operation, such as unexpectedly changed data. (The memory devices in the Apple IIc are especially sensitive to heat.) Letting the machine cool down by turning it off for a while and unblocking the vents before using it again will

bring it back to normal operation. The only exception to this is if you have gotten your Apple IIc *too* hot and physically damaged some internal component.

Disks are another heat-sensitive element of the system. If the built-in drive becomes too hot, a disk within can warp or even melt. A melted or warped disk can't be used again.

11.2 Power Requirements

The electrical power used by the Apple IIc—and everything that draws power from it—is limited by the capacities of the computer's power supply and internal voltage converter. This section describes these limits for the USA external power supply. Appendix G describes them for models built for other countries. The internal voltage converter is the same on all models.

11.2.1 The External Power Supply

If you purchased your Apple IIc outside the USA, consult Appendix G for external power supply characteristics.

The external power supply operates on normal household AC power and provides DC power to the Apple IIc internal converter. The basic specifications of the external power supply are listed in Table 11-2. The Apple IIc external power supply's cord must be plugged into a three-wire 115-volt (nominal) outlet. A two-wire outlet is not properly grounded—using it will damage the external power supply and perhaps the Apple IIc as well. The line voltage must be in the range given in Table 11-2.

▲Warning

Important Safety Instructions: This product is equipped with a three-wire grounding-type plug—a plug having a third (grounding) pin. This plug will only fit into a grounding-type AC outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

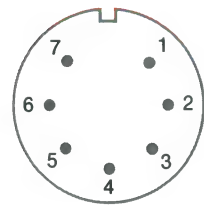
Table 11-2. Power Supply Specifications

Line voltage:	105 to 129 VAC, 60 Hz
Maximum input power consumption:	25 W
Supply voltage:	+15 VDC (nominal)
Supply current:	1.2 A (nominal)

11.2.2 The External Power Connector

The external power supply is attached to the internal converter by means of a 7-pin DIN connector. The connector pins are identified in Figure 11-1 and Table 11-3.

Figure 11-1. External Power Connector



Pin	Signal
1	Not connected
2, 3	Signal ground
4	Shield ground
5, 6	+15 VDC
7	Not connected

Table 11-3. External Power Connector Signals

Pin #	Name	Description
1, 7		Not connected
2, 3	Ground	Common electrical ground
4	Chassis	Chassis ground
5, 6	+15V	+15-volt DC input to converter

11.2.3 The Internal Converter

The internal converter in the Apple IIc operates with a supply voltage from 9 to 20 volts DC as provided by the external power supply or its equivalent. The internal converter provides enough low-voltage electrical power for the built-in electronics plus an external disk drive attached via the 19-pin connector. The basic specifications of the internal converter are listed in Table 11-4. Minus 5 volts is derived from the -12 volts (nominal) provided by the voltage converter.

Table 11-4. Internal Converter Specifications

Input voltage:	+9 to 20 VDC	
Maximum power consumption:	25 W	
Supply voltages:	+5V	$\pm 5\%$
	+12V	$\pm 10\%$
	-12 V	$\pm 10\%$
Maximum supply currents:	+5V:	1.5 A
	+12V:	0.6 A continuous
		0.9 A intermittent
		1.5 A surge (for < 100 ms)
	-12 V:	100 mA
Maximum case temperature:	(-5 V:	50 mA)
	60° C	(140° F)

The Apple IIc uses a switching-type internal voltage converter as a power supply. It is small and lightweight, and it generates less heat than other types of voltage converters.

The voltage converter works by using the DC voltage input to power a variable-frequency oscillator. The oscillator drives a small transformer with several separate windings to produce the different voltages required. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the oscillator circuit. The oscillator circuit uses the error signal to control the duty cycle of its oscillation and keep the output voltages in their normal ranges.

The converter includes circuitry to protect itself and the other electronic parts of the Apple IIc by limiting all three output voltages whenever it detects one of the following malfunctions:

- any supply voltage short-circuited to ground
- any output voltage outside the normal range

Whenever one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, and all the output voltages drop to 0 if they cannot be brought back into their normal range.

11.3 Apple IIc Overall Block Diagram

Figure 11-2 is an overall block diagram of the Apple IIc. The following sections contain more detailed diagrams of the major parts of the machine. A full set of schematic diagrams of the Apple IIc appears in Section 11.14.

11.4 The 65C02 Microprocessor

CMOS (complementary metal-oxide semiconductor) is a way of making integrated circuits that require less power to operate than other technologies such as NMOS (negative-doped metal-oxide semiconductor), used by the 6502.

These instructions are described in Appendix A.

The Apple IIc uses a CMOS 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIc runs at a clock rate of 1.023 MHz and performs up to 500,000 8-bit operations per second.

Note: The clock rate is not a very good criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1-MHz clock is equivalent to many other types of microprocessors with clock rates up to 5 MHz.

In addition to requiring less power than earlier NMOS 6502 processors, the 65C02 in the Apple IIc has 27 new instructions. However, programs that use these additional instructions are not backward compatible with other Apple II series computers that are not equipped with a CMOS 6502.

11.4.1 65C02 Block Diagram

Figure 11-3 is a block diagram of the 65C02 microprocessor. Table 11-5 contains the general specifications of this chip. The 65C02 has a 16-bit address bus, giving it an address space of 64K bytes. The Apple IIc uses special techniques to address a total of more than 64K (see Chapter 2).

Figure 11-2. Apple IIc Block Diagram

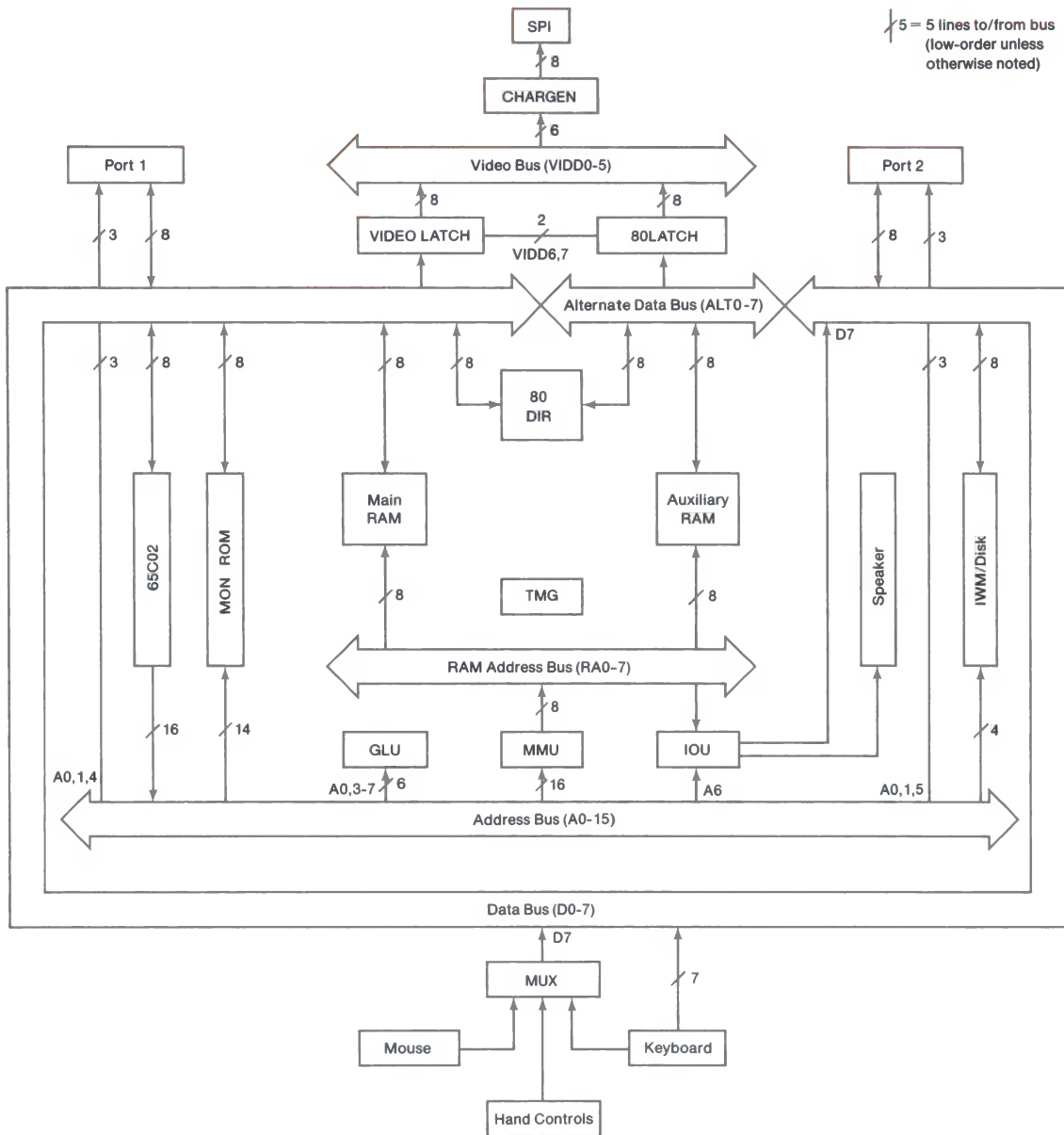


Figure 11-3. 65C02 Block Diagram

Copyright 1982, NCR Corporation. Used by permission of NCR Corporation, Dayton, Ohio.

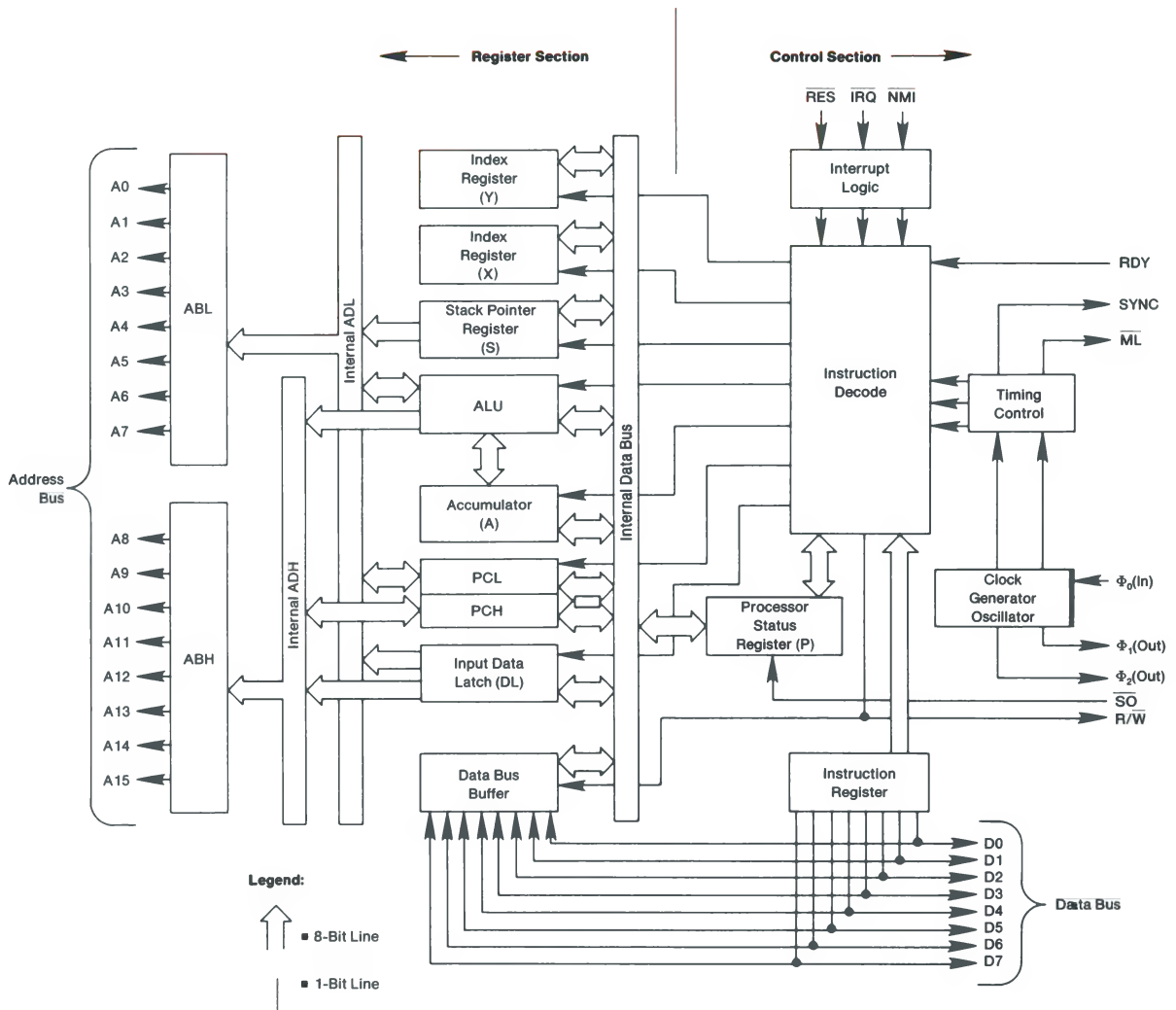


Table 11-5. 65C02 Microprocessor Specifications

Type:	65C02
Register complement:	8-bit accumulator (A) 8-bit index registers (X,Y) 8-bit stack pointer (S) 8-bit processor status (P) 16-bit program counter (PC)
Data bus:	8 bits wide
Address bus:	16 bits wide
Address range:	65,536 (64K)
Interrupts:	IRQ (maskable) NMI (nonmaskable) BRK (programmed)
Operating voltage:	+5V ($\pm 5\%$)
Power dissipation:	5 mW (at 1 MHz)

11.4.2 65C02 Timing

The Apple IIc's operation is controlled by a set of synchronous timing signals, sometimes called clock signals. The Apple IIc uses a 14.318-MHz master timing signal, called 14M, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the 65C02's operation are described in this section. Other timing signals are described in Sections 11.6.2, 11.9.3, and 11.9.4.

The relationships of the main 65C02 timing signals are diagrammed in Figure 11-4, and the signals are listed in Table 11-6. The 65C02 clock signals are $\phi 1$ and $\phi 0$, complementary signals at a frequency of 1.0227 MHz. The Apple IIc signal named $\phi 0$ is similar to the signal called $\phi 2$ in Appendix A (it isn't identical—it's a tiny bit early).

Figure 11-4. 65C02 Timing Signals

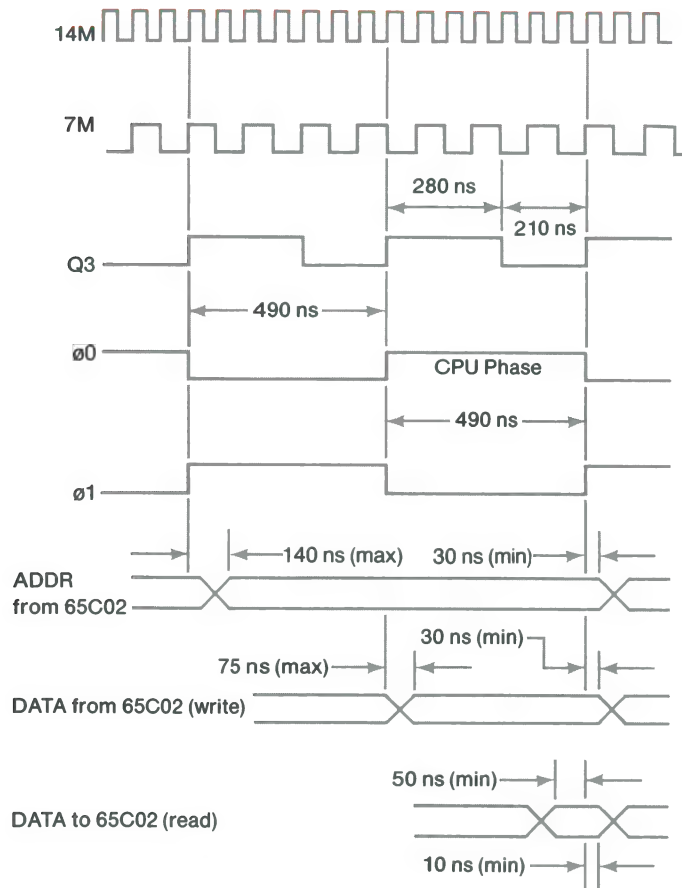


Table 11-6. 65C02 Timing Signal Descriptions

Signal Name	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
$\phi 0$	Phase 0 of 65C02 clock, 1.0227 MHz; complement of $\phi 1$
$\phi 1$	Phase 1 of 65C02 clock, 1.0227 MHz; complement of $\phi 0$

The 65C02's operations are related to the clock signals in a simple way: internal during $\phi 1$, external during $\phi 0$. The 65C02 puts an address on the address bus during $\phi 1$. This address is valid not later than 110 nanoseconds after $\phi 1$ goes high and remains valid through all of $\phi 0$. The 65C02 reads or writes data during $\phi 0$. If the 65C02 is writing, the read/write signal is low during $\phi 0$ and the 65C02 puts data on the data bus. The data are valid not later than 75 nanoseconds after $\phi 0$ goes high. If the 65C02 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of $\phi 0$.

More information about the 65C02 and its instruction set is in Appendix A.

11.5 The Custom Integrated Circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIc is in five custom integrated circuits:

- the memory management unit (MMU)
- the input-output unit (IOU)
- the timing generator (TMG)
- the general logic unit (GLU)
- the disk controller unit, also known as the Integrated Woz Machine (IWM)

The soft switches that control the various I/O and addressing modes of the Apple IIc are addressable flags inside the MMU, IOU, and GLU. The functions of the MMU and IOU are not as independent as their names suggest; working together, they generate all the addressing signals. For example, the MMU generates the RAM address signals for the CPU, while the IOU generates similar RAM address signals for the video display and most I/O hardware addresses.

11.5.1 The Memory Management Unit (MMU)

The circuitry inside the MMU implements these soft switches:

- Page 2 display (Page2) (described in Chapter 5)
- high-resolution mode (HiRes) (Chapter 5)
- store to 80-column display (80Store) (Chapter 5)
- select bank 2 (Bank2) (Chapter 2)
- enable bank-switched RAM (EnlCRAM) (Chapter 2)

- read auxiliary memory (RAMRd) (Chapter 2)
- write auxiliary memory (RAMWrt) (Chapter 2)
- auxiliary stack and zero page (AltZP) (Chapter 2)
- reset mouse Y interrupt (RstYInt) (Chapter 9)
- reset mouse X interrupt (RstXInt) (Chapter 9)

These switches are available on MMU pin 21, which is connected to bit 7 on the data bus. Figure 11-5 shows the MMU pinouts; Table 11-7 describes the signals.

Important!

A signal name followed by an asterisk is active low—that is, it is true when the signal is at a TTL high (+5V) level.

The 64K dynamic RAMs used in the Apple IIc use a multiplexed address, as described in Section 11.6.2. The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.

Figure 11-5. The MMU Pinouts

GND	1	40	A1
A0	2	39	A2
$\phi 0$	3	38	A3
Q3	4	37	A4
PRAS*	5	36	A5
RA0	6	35	A6
RA1	7	34	A7
RA2	8	33	A8
RA3	9	32	A9
RA4	10	31	A10
RA5	11	30	A11
RA6	12	29	A12
RA7	13	28	A13
R/W*	14	27	A14
INH*	15	26	A15
C06X*	16	25	+5V
EN80*	17	24	SELIO*
KBD*	18	23	CASEN*
ROMEN2*	19	22	C07X*
ROMEN1*	20	21	MD7

Table 11-7. The MMU Signal Descriptions

Pin #	Name	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	$\phi 0$	Clock phase 0 input
4	Q3	Timing signal input
5	PRAS*	Memory row-address strobe
6–13	RA0–RA7	Multiplexed address output
14	R/W*	65C02 read-write control signal
15	INH*	Inhibits main memory (tied to +5V)
16	C06X*	Causes \$C06x outputs to go to 0 during $\phi 0$
17	EN80*	Enables auxiliary RAM
18	KBD*	Enables keyboard data bits 0–6
19	ROMEN2*	Enables ROM (tied to ROMEN1*)
20	ROMEN1*	Enables ROM (tied to ROMEN2*)
21	MD7	State of MMU flags on data bus bit 7
22	C07X	Causes \$C07x outputs to go to 0 during $\phi 0$
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during $\phi 0$ for any access to \$C0 page except \$C08x, Bx, Cx, or Fx
25	+5V	Power
26–40	A15–A1	65C02 address input

11.5.2 The Input/Output Unit (IOU)

Input/output unit (IOU) implements the following soft switches, all described in Chapters 2 and 3:

- ☐ Page 2 display (Page2)
- ☐ high-resolution mode (HiRes)
- ☐ text mode (TEXT)
- ☐ mixed mode (MIXED)
- ☐ 80-column display (80Col)
- ☐ character-set select (AltChar)
- ☐ any-key-down (AKD)
- ☐ mouse movement (X0, Y0)
- ☐ vertical blanking interrupt (VBIInt)

These switches are available on IOU pin 9, which is connected to bit 7 on the data bus. Figure 11-6 shows the IOU pinouts; Table 11-8 describes the signals.

The 64K dynamic RAMs used in the Apple IIc require a multiplexed address, as described in Section 11.6.2. The IOU generates this multiplexed address during clock phase 1 for the data transfers required for display and memory refresh. The way this address is generated is described in Section 11.9.1.

Figure 11-6. The IOU Pinouts

GND	1	40	H0
GR	2	39	SYNC*
SEGA	3	38	WNDW*
SEGB	4	37	CLRGAT*
VC	5	36	RA10*
80COL*	6	35	RA9*
CASSO	7	34	VIDD6
SPKR	8	33	VIDD7
MD7	9	32	KSTRB
YMOVE	10	31	AKD
(N.C.)	11	30	IOUSELIO*
(N.C.)	12	29	A6
PDL0/XMOVE	13	28	+5V
R/W*	14	27	Q3
RESET*	15	26	ø0
IRQ*	16	25	PRAS*
RA0	17	24	RA7
RA1	18	23	RA6
RA2	19	22	RA5
RA3	20	21	RA4

Table 11-8. The IOU Signal Descriptions

Pin #	Name	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects high-resolution when low, low-resolution when high
5	VC	Displays vertical counter bit: in text mode, SEGA, SEGB, and VC determine which of the eight rows of a character's dot pattern to display; in low-resolution, selects upper or lower block defined by a byte

Table 11-8—continued. The IOU Signal Descriptions

Pin #	Name	Description
6	80COL*	80-column video enable
7	CASSO	Reserved
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	YMOVE	Detects mouse movement along Y axis
11	N.C.	Not used
12	N.C.	Not used
13	PDL0/XMOVE	Detects mouse movement along X axis
14	R/W*	65C02 read-write control signal
15	RESET*	Power on and reset output
16	IRQ*	Maskable interrupt line to 65C02
17–24	RA0–RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS*	Row-address strobe (phase 0)
26	$\phi 0$	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	IOUSELIO*	Derived from the SELIO* output for MMU pin 24
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33, 34	VIDD7, VIDD6	Video display data bits
35, 36	RA9*, RA10*	Video display control bits
37	CLRGAT*	Color-burst gate (enable)
38	WNDW*	Displays blanking signal
39	SYNC*	Displays synchronization signal
40	H0	Displays horizontal timing signal (low bit of character counter)

11.5.3 The Timing Generator (TMG)

A custom timing generator chip (TMG) generates several timing and control signals in the Apple IIc. The TMG pinouts are shown in Figure 11-7; the signals are listed in Table 11-9.

Figure 11-7. The TMG Pinouts

14M	1	20	+5V
7M	2	19	PRAS*
CREF	3	18	(N.C.)
H0	4	17	PCAS*
VIDD7	5	16	Q3
SEGB	6	15	$\phi 0$
TEXT	7	14	$\phi 1$
CASEN*	8	13	VID7M
80COL*	9	12	LDPS*
GND	10	11	TMGEN*

Table 11-9. The TMG Signal Descriptions

Pin #	Name	Description
1	14M	14.318-MHz master timing signal input
2	7M	7.159-MHz timing signal
3	CREF	3.5795-MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VIDD7	Video data bit 7
6	SEGB	Video timing signal
7	TEXT	Video display text-modes enable
8	CASEN*	RAM enable (CAS enable)
9	80COL*	Enables 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enables master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	$\phi 1$	Phase 1 system clock
15	$\phi 0$	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS*	RAM column-address strobe
18	N.C.	Reserved for testing
19	PRAS*	RAM row-address strobe
20	+5V	Power

11.5.4 The General Logic Unit (GLU)

The general logic unit is a single chip that contains the miscellaneous logic required for the system. It provides

- all RAM read/write timing
- double-high-resolution enable/disable
- soft-switch status registers
- write command registers
- IOU control for mouse interrupts
- double-high-resolution soft switches.

The GLU's pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

Figure 11-8. The GLU Pinouts

14M	1	24	+5V
A0	2	23	SER*
A3	3	22	IOUHOLE
A4	4	21	DISK*
A5	5	20	7M
A6	6	19	CREF
A7	7	18	(N.C.)
ø0	8	17	(N.C.)
SELIO*	9	16	TEXT
GR	10	15	R/W*
RESET*	11	14	MD7
GND	12	13	GLUEN*

Table 11-10. The GLU Signal Descriptions

Pin #	Name	Description
1	14M	Master clock (14.318 MHz)
2, 3-7	A0, A3-A7	Address lines to select least significant byte of addresses on C0 page
8	ø0	Phase 0 of 1.0227-MHz processor sync clock
9	SELIO*	Device select for selecting most significant byte of the address
10	GR	Graphics mode select line
11	RESET*	Master reset for system; resets GLU
12	GND	Ground reference and negative supply
13	GLUEN*	Enables GLU
14	MD7	Indicates status of MMU flags on data bus bit 7
15	R/W*	Read/write qualifier input from processor
16	TEXT	Signal used to generate video timing in double-high-resolution or not-graphics
17, 18	N.C.	Not used
19	CREF	Color reference signal
20	7M	7-MHz clock output
21	DISK*	Disk controller device select output

Table 11-10—continued. The GLU Signal Descriptions

Pin #	Name	Description
22	IOUHOLE	Controls IOUSELIO
23	SER*	Serial controller device select output
24	+5V	+5 volt supply

11.5.5 The Disk Controller Unit (IWM)

For further information on group code recording, refer to Section 11.10.

The IWM (for Integrated Woz Machine) is a disk controller that includes, on a single chip, all the capabilities of the disk controller card originally designed by Steve Wozniak in 1977.

Right after reset, the IWM is an integrated GCR (group code recording) disk drive controller. It also has a status register, mode register, and multiple operating modes. It provides both synchronous and asynchronous modes, and a fast mode with a data rate twice that of normal disk I/O speeds. Figure 11-9 shows the IWM pin assignments; Table 11-11 describes the IWM signals.

Figure 11-9. The IWM Pinouts

SEEKPH0	1	28	SEEKPH1
SEEKPH2	2	27	SEEKPH3
A0	3	26	+5V
A1	4	25	Q3
A2	5	24	7M
A3	6	23	RESET*
DISK*	7	22	RDDATA
WRDATA	8	21	WRPROT
WRREQ*	9	20	DR1*
D0	10	19	DR2*
D1	11	18	D7
D2	12	17	D6
D3	13	16	D5
GND	14	15	D4

Table 11-11. The IWM Signal Descriptions

Pin #	Name	Description
1	SEEKPH0	Stepper motor control phase 0, one of four programmable disk drive motor phase outputs
2	SEEKPH2	Stepper motor control phase 2
3	A0	The data input to the state bit selected by A1 to A3
4-6	A1-A3	These three inputs select one of the eight bits in the state register to be updated.
7	DISK*	Device enable. The falling edge of DISK* latches information on A1 to A3. The rising edge of either Q3 or DISK* qualifies write register data.
8	WRDATA	The serial data output. Each 1-bit causes a transition on this output.
9	WRREQ*	This signal is a programmable buffered output line.
10-13	D0-D3	D0 to D7 make up the bidirectional data bus.

Table 11-11—continued. The IWM Signal Descriptions

Pin #	Name	Description
14	GND	Ground reference and negative supply
15–18	D4–D7	The remaining bits of the bidirectional data bus
19	DR2*	Drive 2 select
20	DR1*	Drive 1 select
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register.
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to 0.
24	7M	7-MHz clock input
25	Q3	A 2.0-MHz clock input used to qualify the timing of the serial data being written or read
26	+5V	The +5 volt supply
27	SEEKPH3	Stepper motor control phase 3
28	SEEKPH1	Stepper motor control phase 1

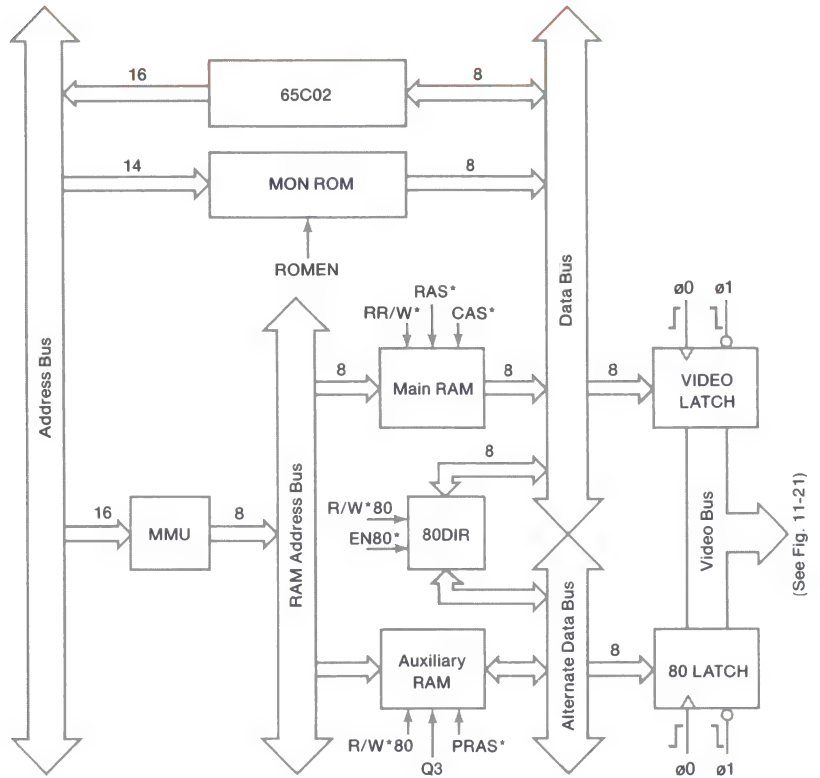
11.6 Memory Addressing

The 65C02 microprocessor can directly address 65,536 locations. The Apple IIc uses this entire address space, and then some: some are as in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIc and the way they are addressed. Input and output also use portions of the memory address space; refer to Chapter 2 for information.

Figure 11-10 illustrates the Apple IIc's overall memory bus organization and memory selection signals.

Note: Some Apple IIc's have ROMs with 27xx designations, some have 23xx. They are functionally equivalent.

Figure 11-10. Memory Bus Organization



11.6.1 ROM Addressing

In the Apple IIc the following programs are permanently stored in a type 23128 16K-by-8-bit ROM (Figure 11-11):

- ☐ Applesoft editor and interpreter
- ☐ Monitor
- ☐ enhanced video firmware

UniDisk 3.5

The version of the Apple IIc that supports the UniDisk 3.5 uses a 23256 32K-by-8-bit ROM. It needs the extra space for the Protocol Converter, Mini-Assembler, and other added functions that it supports.

Figure 11-11. The 23128 ROM Pinouts
Note: In the type 23256 ROM, pin #27 is A14.

+5V	1	28	+5V
A12	2	27	(N.C.)
A7	3	26	A13
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	OE*
A2	8	21	A10
A1	9	20	CE*
A0	10	19	D7
D0	11	18	D6
D1	12	17	D5
D2	13	16	D4
GND	14	15	D3

Figure 11-12. The 2316 ROM Pinouts

KA7	1	24	+5V
KA6	2	23	KA8
KA5	3	22	CAPS
KA4	4	21	+5V
KA3	5	20	KBD*
KA2	6	19	LANGSW
KA1	7	18	GND
KA0	8	17	(N.C.)
D0	9	16	D6
D1	10	15	D5
D2	11	14	D4
GND	12	13	D3

Figure 11-13. The 2364 ROM Pinouts

+5V	1	28	+5V
A12	2	27	+5V
A7	3	26	+5V
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	GND
A2	8	21	A10
A1	9	20	WNDW*
A0	10	19	O7
O0	11	18	O6
O1	12	17	O5
O2	13	16	O4
GND	14	15	O3

The ROM is enabled by two signals called ROMEN1 and ROMEN2. (In the Apple IIc, ROMEN1 and ROMEN2 are electrically connected.) The segment of the ROM enabled by ROMEN1 occupies the memory address space from \$C100 through \$DFFF. The address space from \$C300 through \$C3FF and much of \$C800 through \$CFFF contains the enhanced video firmware.

These ROM address allocations are approximately true (some space sharing takes place):

- ROM addresses \$C000 through \$C0FF are never available.
- ROM addresses \$C100 and \$C200 are entry points to firmware for serial ports 1 and 2, respectively.
- ROM address \$C400 is the entry point to mouse interface support.
- ROM addresses \$C500 through \$C5FF are reserved.
- ROM address \$C600 is the entry point to firmware for the built-in and external disk drives. The built-in drive is considered slot 6 drive 1 or its equivalent. The external drive is considered slot 6 drive 2.
- ROM addresses starting at \$C700 support (from the Monitor) the external drive as if it were slot 7 drive 1, for external-drive startup only.
- Addresses \$D000 through \$F7FF contain the Applesoft BASIC interpreter; addresses \$F800 through \$FFFF contain the Monitor firmware.

The other ROMs in the Apple IIc are a type 2316 ROM (Figure 11-12) used for the keyboard character decoder, and a type 2364 ROM (Figure 11-13) used for character sets for the video display. This 2364 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

11.6.2 RAM Addressing

The RAM (programmable) memory in the Apple IIc is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIc, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIc take advantage of the two-phase system clock described in Section 11.4.2 to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during $\phi 0$, and the display circuits read data only during $\phi 1$.

Dynamic RAM Refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIc reads the data in the active display page and sends them to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIc refreshes the display 60 times per second.

The dynamic RAM devices used in the Apple IIc also need a kind of refreshment, because the data are stored in the form of electric charges that diminish with time and must be replenished. The Apple IIc is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called multiplexing. Since only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs (Figure 11-14).

Figure 11-14. The 64K RAM Pinouts

+5V	1	16	GND
MDx	2	15	CAS*
R/W*	3	14	MDx
RAS*	4	13	RA1
RA7	5	12	RA4
RA5	6	11	RA3
RA6	7	10	RA2
+5V	8	9	RA0

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described in Section 11.9.1, the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated 8 times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every 2 milliseconds (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

Table 11-12. RAM Address Multiplexing

Mux'd Address	Row Address	Column Address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Dynamic RAM Timing

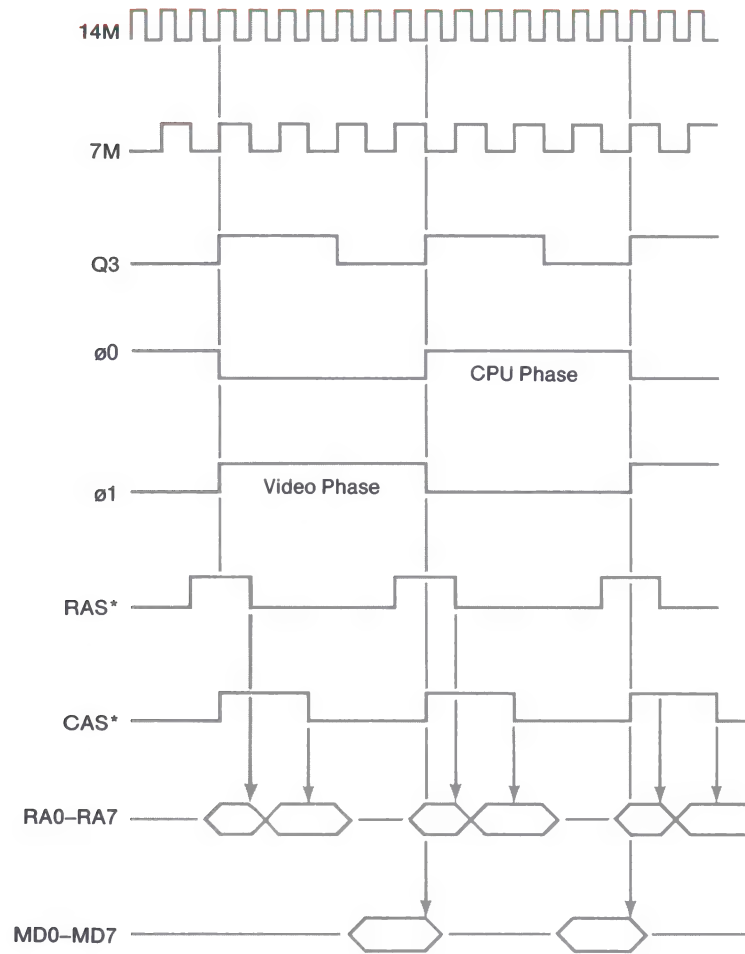
The Apple IIc's microprocessor clock runs at a speed of 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2-MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU are strobed by the falling edge of $\phi 0$, and display data are strobed by the falling edge of $\phi 1$, as shown in Figure 11-15.

The RAM timing looks complicated because the RAM address is multiplexed, as described in the previous section. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines labeled RA0–RA7 (Table 11-13). Along with the other timing signals, the TMG generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).

Table 11-13. RAM Timing Signals

Signal Name	Description
$\phi 0$	Clock phase 0 (CPU phase)
$\phi 1$	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/column-address strobe
RA0–RA7	Multiplexed address bus
MD0–MD7	Internal data bus

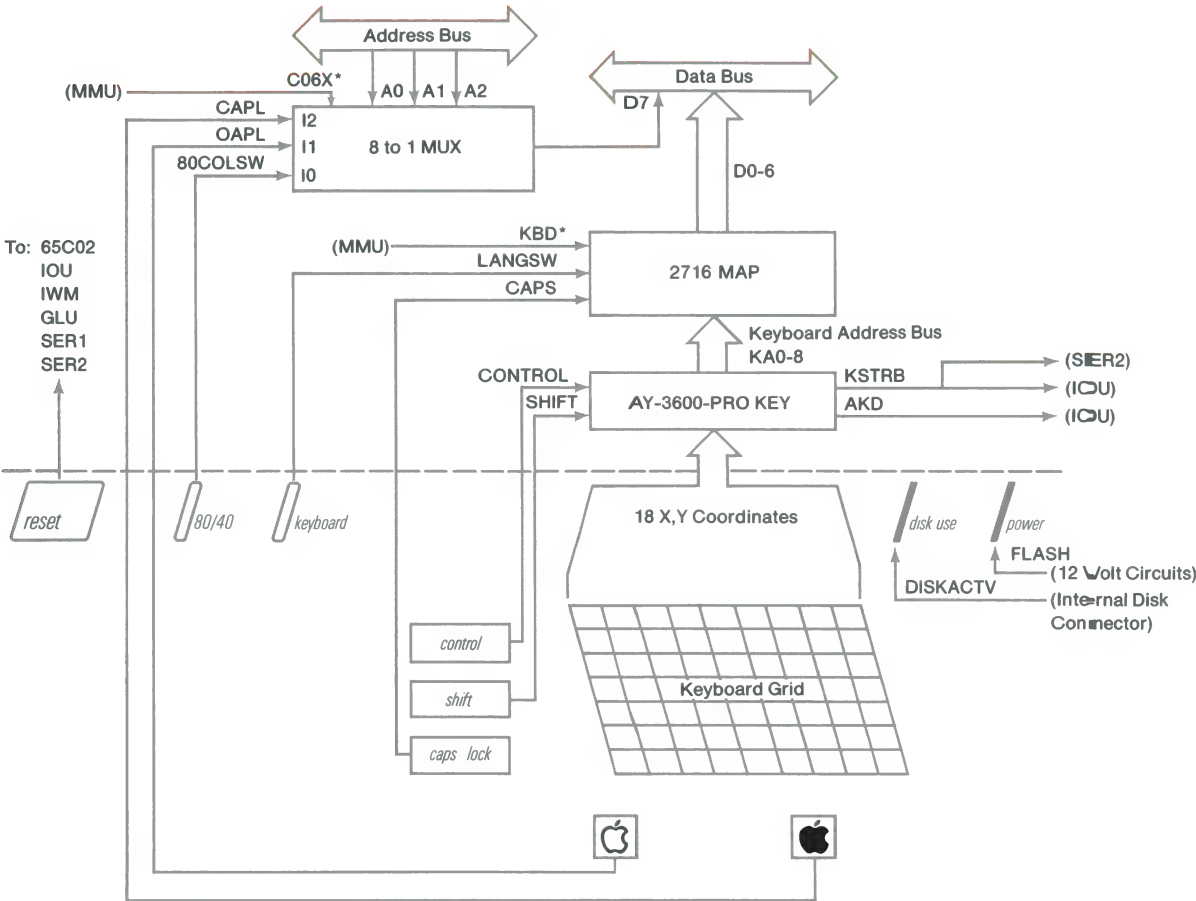
Figure 11-15. RAM Timing Signals



11.7 The Keyboard

The Apple IIc's keyboard is a matrix of keyswitches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector (Figure 11-16). The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C46 and R6. The debounce time is also set externally, by C45.

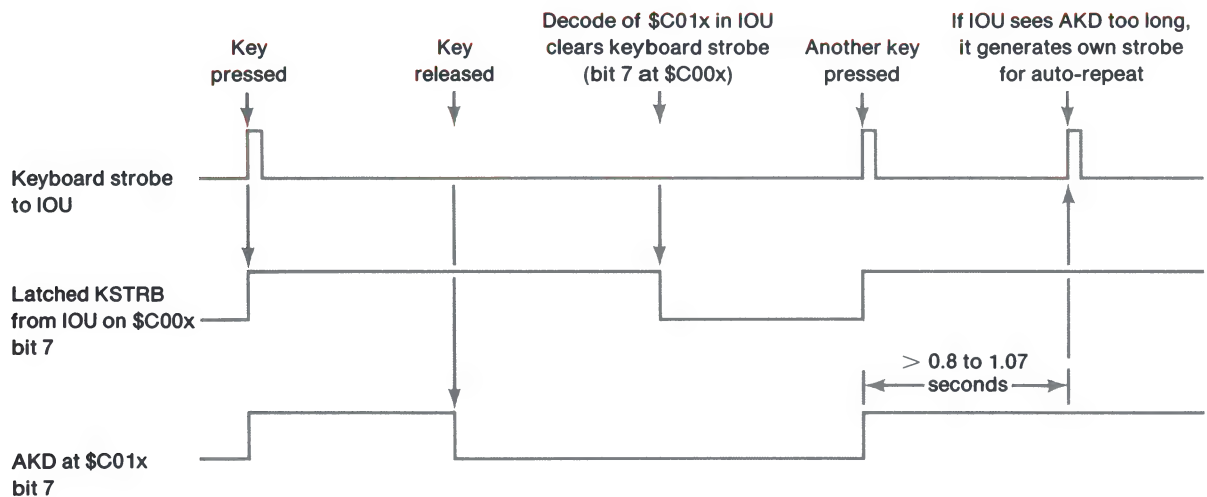
Figure 11-16. Keyboard Circuit Diagram



The AY-3600's outputs include five bits of key code plus separate lines for CONTROL, SHIFT, any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code line and CONTROL and SHIFT are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named KBD* and ENKBD*. The KBD* signal is enabled by the MMU whenever a program reads location \$C000, as described in Chapter 2.

Figure 11-17 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

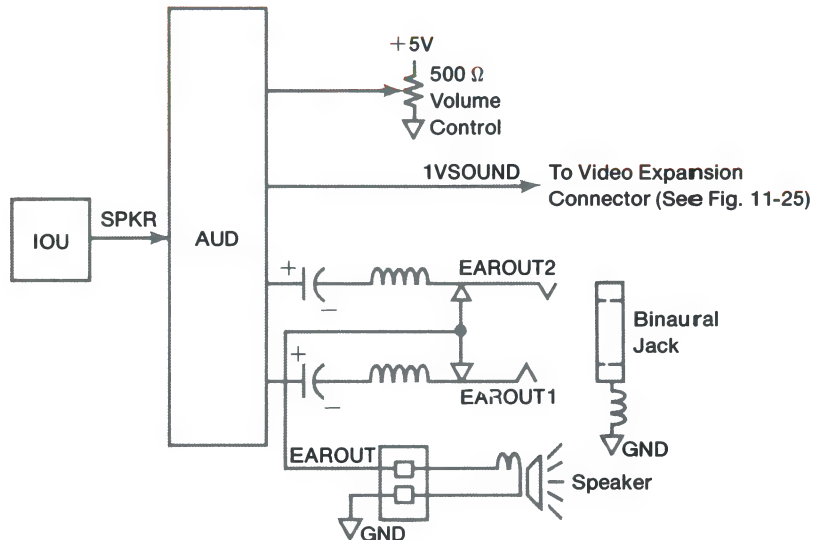
Figure 11-17. Keyboard Signals



11.8 The Speaker

The Apple IIc's built-in loudspeaker is controlled by a single bit of output from the input/output unit (IOU), amplified by a hybrid circuit (Figure 11-18).

Figure 11-18. Speaker Circuit Diagram



11.8.1 Volume Control

AUD is an audio-amplifier hybrid circuit.

There is a 500-ohm variable resistor feeding anywhere from 0 to 5 volts to pin 5 of AUD to control the speaker volume. This potentiometer controls the volume of both the built-in speaker and whatever is plugged into the output jack.

11.8.2 Output Jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5-mm audio output jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural), providing sound to both channels. Inserting a headphone plug into the jack disconnects the built-in speaker.

11.9 The Video Display

The Apple IIc produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that are being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

Note: Apple IIc computers manufactured for sale in the USA generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIc's used in European countries require an external adapter to provide video that is compatible with the standard used there, which is called PAL (for phase alternating lines). References to the PAL standard are found in the bibliography at the end of this manual. This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named WNDW* is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the blanking intervals, the display is blank and the WNDW* signal is high. The synchronization signals, called sync for short, are produced by making the signal named SYNC* low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

11.9.1 The Video Counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE*. The input to the horizontal counter is the 1-MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count from 0 to 64, then start over at 0. Whenever this happens, HPE* forces another count with H0 through H5 held at 0, extending the total count to 65.

The IOU uses the 40 horizontal count values from 25 through 64 in generating the low-order part of the display data address, as described in Section 11.9.3. The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from 0. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 262 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIc's video display is not interlaced.)

11.9.2 Display Memory Addressing

As described in Section 5.7, data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data are stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal \$0400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing them directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary.

The address transformation that folds three rows of 40 display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described in Section 11.9.4.

11.9.3 Display Address Mapping

Consider the simplest display on the Apple IIc, the 40-column text mode. To address 40 columns requires 6 bits, and to address 24 rows requires another 5 bits, for a total of 11 address bits. Addressing the display this way would involve 2048 (2 to the 11th power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to

display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- map the 960 bytes of 40-column text into only 1024 bytes
- scan the low-order address to refresh the dynamic RAMs
- continue to refresh the RAMs during video blanking

The requirements for RAM refreshing are discussed in Section 11.6.2.

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals called S0, S1, S2, and S3, where S stands for sum. Figure 11-19 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5*. A constant value of 1 appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is 0 and the horizontal count is 24: H0, H1, H2, and H5 are 0s, and H3 and H4 are 1s. The value of the sum is 0, so the memory location for the first character on the display is the first location in the display page, as you might expect.

Figure 11-19. Display Address Transformation

		V3		Carry in
H5*	V3	H4	H3	Augend
V4	H5*	V4	1	Addend
S3	S2	S1	S0	Sum

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 11-14). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from 0 to 4 and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 11-14, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are 0 again, and V3 changes to 1. If you do the addition in Figure 11-19 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

Table 11-14 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2, and S3 are the folded address bits described above. Table 11-15 shows memory address bits for the display modes.

Table 11-14. Display Memory Addressing

Memory Address Bit	Display Address Bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	*
A11	*
A12	*
A13	*
A14	*
A15	GND

* For these address bits, see Table 11-15.

Table 11-15. Memory Address Bits for Display Modes

Note: . means logical AND; ' means logical NOT.

Address Bit	Display Modes	
	Text and Low-Resolution	High-Resolution and Double-High-Resolution
A10	80STORE+PAGE2'	VA
A11	80STORE'.PAGE2	VB
A12	0	VC
A13	0	80STORE+PAGE2'
A14	0	80STORE'.PAGE2

Figure 11-20 shows how groups of three 40-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 5-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

Figure 11-20. 40-Column Text Display Memory

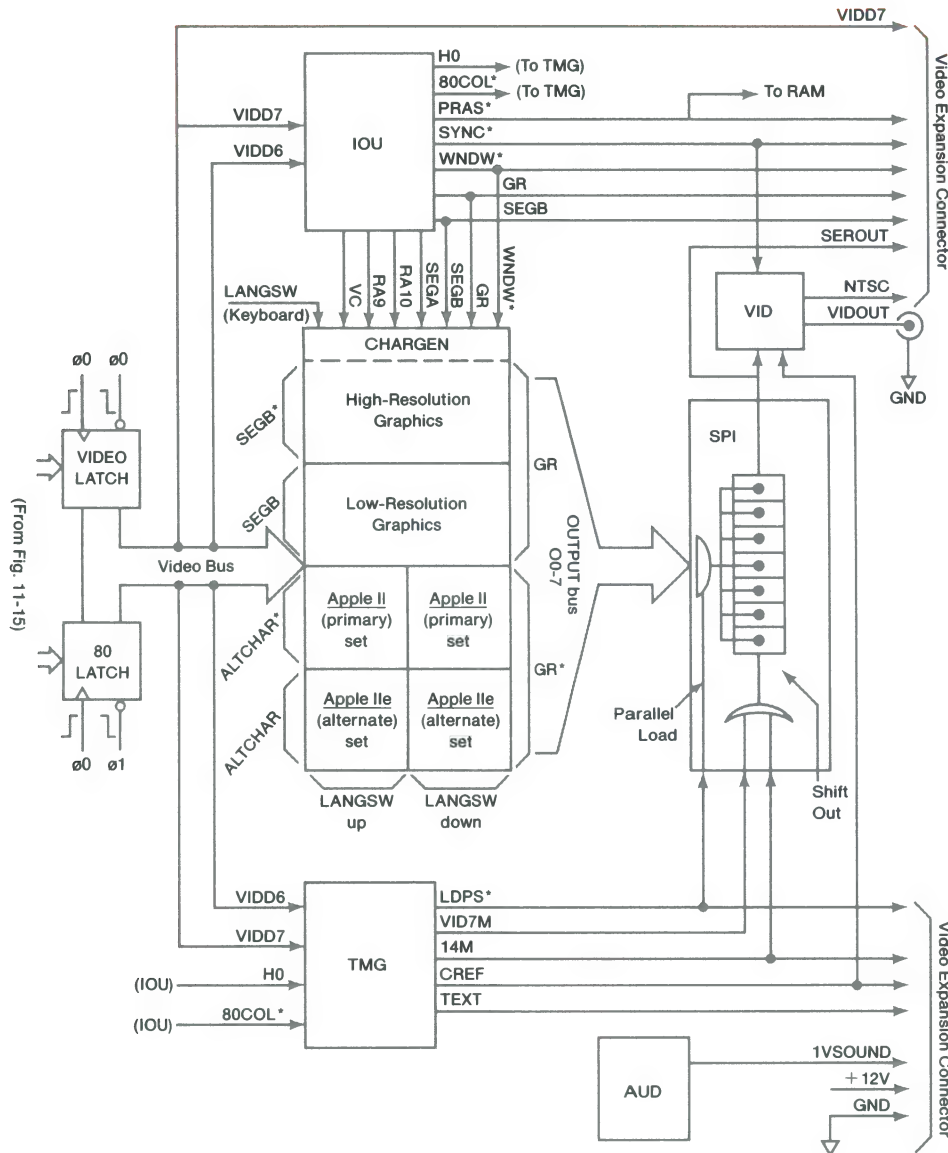
Note: Memory locations marked with a double asterisk (**) are screen holes, described in Section 2.5.1.

	128 Bytes			
	40 Bytes	40 Bytes	40 Bytes	8 Bytes
\$400	Row 0	Row 8	Row 16	**
\$480	Row 1	Row 9	Row 17	**
\$500	Row 2	Row 10	Row 18	**
\$580	Row 3	Row 11	Row 19	**
\$600	Row 4	Row 12	Row 20	**
\$680	Row 5	Row 13	Row 21	**
\$700	Row 6	Row 14	Row 22	**
\$780	Row 7	Row 15	Row 23	**

11.9.4 Video Display Modes

The different display modes all use the address-mapping scheme described in Section 11.9.3, but they use different-sized memory areas in different locations. This section describes the addressing schemes and the methods of generating the actual video signals for the different display modes. Figure 11-21 illustrates the video display circuits discussed in this section.

Figure 11-21. Video Display Circuits



Text Displays

The text and low-resolution graphics pages begin at memory locations \$0400 and \$0800. Table 11-15 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of Page2 and 80Store, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to 0. Notice that 80Store active inhibits Page2: there is only one display page in 80-column mode.

The low-order six bits of each data byte reach the character generator directly, via the video data bus VID0–VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit (Figure 11-21). The shift register is controlled by signals named LDPS* (for load parallel-to-serial shifter) and VID7M (for video 7 MHz). In 40-column mode, LDPS* strobes the output of the character generator into the shift register once each microsecond, and VID7M shifts the bits out at 7 MHz (Figure 11-22).

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory in auxiliary memory are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0–VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously (at the rising edge of $\phi 0$), but their outputs are sent to the character generator alternately by the falling edge of $\phi 0$ and $\phi 1$. In 80-column mode, LDPS* loads data from the character generator into the shift register twice during each microsecond, once during $\phi 0$ and once during $\phi 1$, and VID7M remains low, enabling the clock continuously at 14M (Figure 11-23).

Figure 11-22. 7-MHz Video Timing Signals (40-Column, Low-Resolution, and High-Resolution Display)

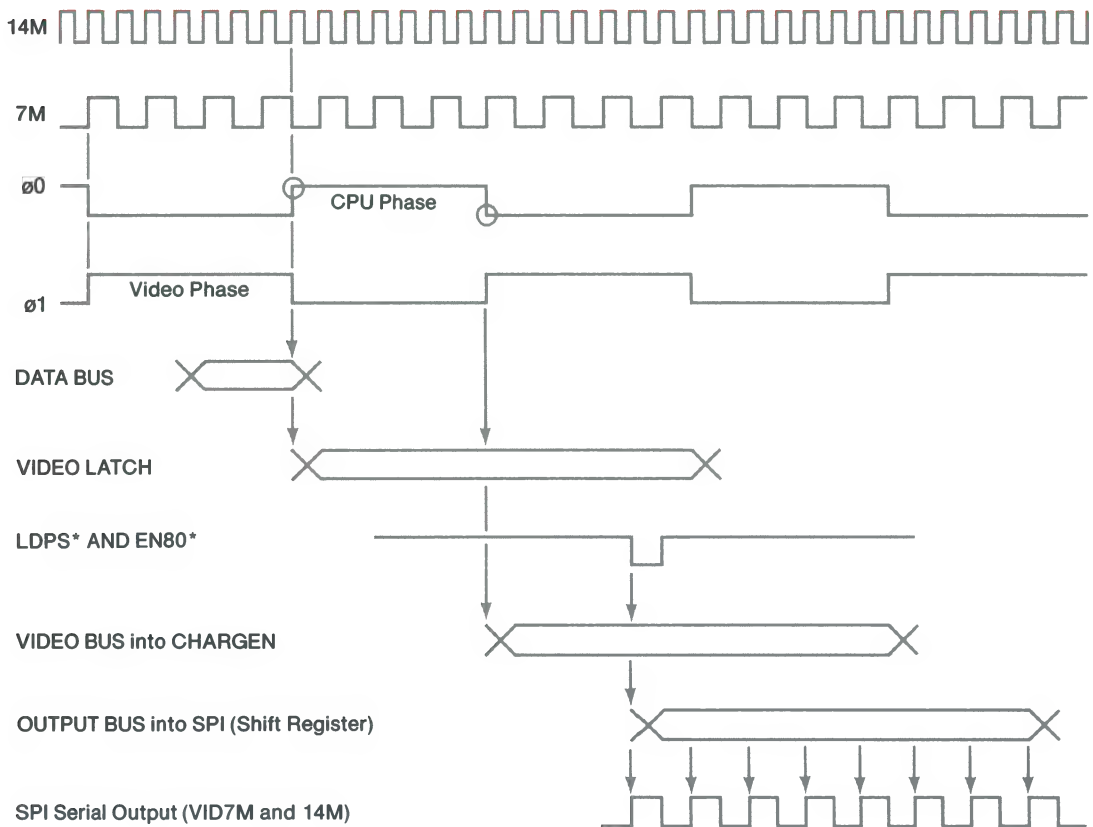
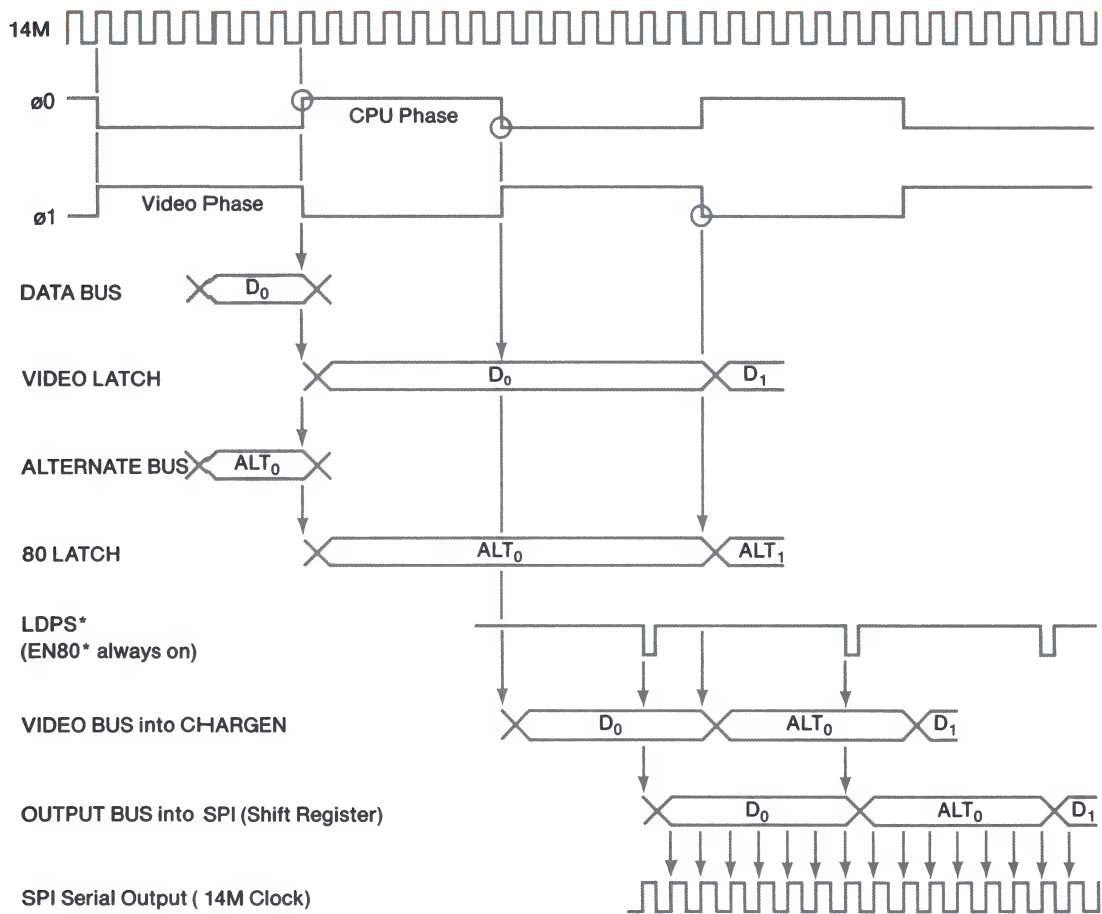


Figure 11-23. 14-MHz Video Timing Signals (80-Column and Double-High-Resolution Display)



Low-Resolution Display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES*, as shown in Table 11-16.

Table 11-16. Character-Generator Control Signals

Display Mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES*	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nibbles. Each nibble selects 1 of 16 colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits (Figure 11-21).

The video signal generated by the Apple IIc includes a short burst of 3.58-MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58-MHz color signal. The Apple IIc's video signal produces color by interacting with this 3.58-MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58-MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1-MHz data clock. To generate a stream of 14 bits from each 8-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same 8 bits; the last 2 bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58-MHz color signal, the phase relationship between the bit

patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

High-Resolution Display

The high-resolution graphics pages begin at memory locations \$2000 and \$4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PAGE2 and 80STORE, the signals controlled by the display-page (Page2) and 80-column-video (80Col) soft switches. As in text mode, 80STORE inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 11-20, but there are eight of these blocks. As Tables 11-14 and 11-15 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display, VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block.

It might help to think of this scheme as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the seven bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0-VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58-MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58-MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating 1s and 0s gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating 1s and 0s.

To produce different colors, the bit patterns must have different phase relationships to the 3.58-MHz color signal. If alternating 1s and 0s produce a certain color, say green, then reversing the pattern to 0s and 1s will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58-MHz color signal. In high-resolution mode, the Apple IIc produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the TMG. If D7 is off, the TMG transmits shift-register timing signals LDPS* and VID7M normally. If D7 is on, the TMG delays LDPS* and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

A Note About Timing: For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the TMG controls shift-register timing signals LDPS* and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

Double-High-Resolution Display

Double-high-resolution graphics mode displays two bytes in the time normally required for one, but it uses high-resolution graphics Pages 1 and 1X instead of text and low-resolution Pages 1 and 1X.

Note: There is a second pair, HRP2 and HRP2X, which can be used to display a second double-high-resolution page.

Double-high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, 7 from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appear in columns 0–6, 14–20, and so on, up to columns 547–552. Data from main memory appear in columns 7–13, 21–27, and so on, up to 553–559.

RGB stands for red, green, and blue, and identifies a type of color monitor that uses independent inputs for the three primary colors.

For further information about double-high-resolution graphics display, refer to the Bibliography.

VID is a video-amplifier hybrid circuit.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth (less than 14 MHz) monitor, single dots are dimmer than normal.

Note: Except for some expensive RGB-type color monitors, any video monitor with a bandwidth as high as 14 MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of $\phi 0$ clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch (Figure 11-21).

$\Phi 1$ enables output from the (auxiliary) 80 latch, and $\phi 0$ enables output from the (main) video latch. Output from both latches goes to CHARGEN, where GR and SEGB* select high-resolution graphics. LDPS operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for double-high-resolution display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gates shift register output to VID, the video display hybrid circuit, for output to the display device.

11.9.5 Video Output Signals

The stream of video data generated by the display circuits described above goes to a hybrid circuit (VID) that adjusts the signals to the proper amplitudes and conditions the color burst.

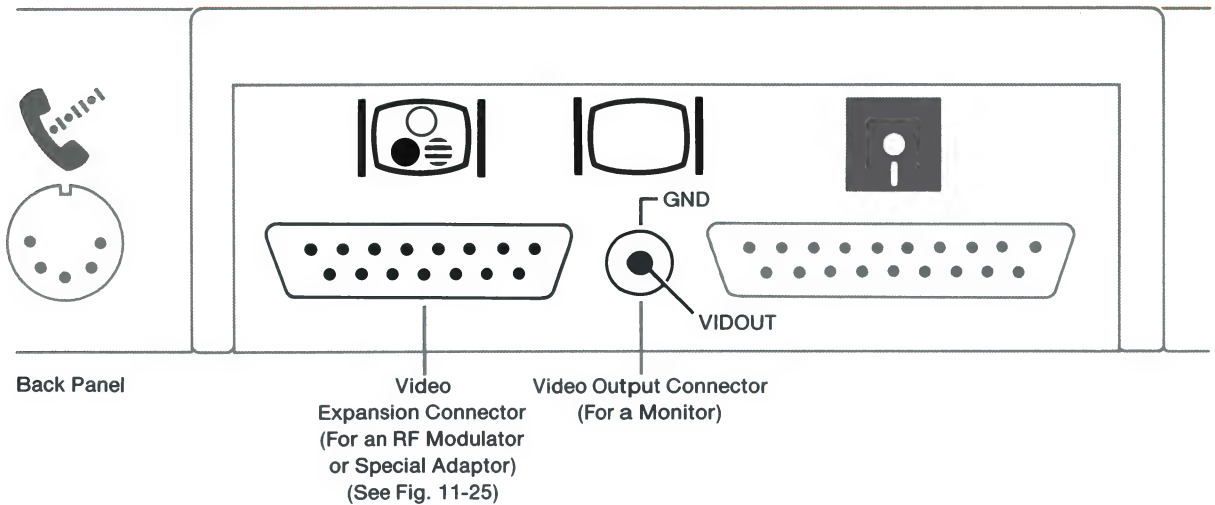
The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video. This signal is available in two places in the Apple IIc (Figure 11-24):

- at the video output connector on the back of the Apple IIc
- at the video expansion connector (pin 12) on the back panel (Table 11-17)

Monitor Output

The sleeve of the video output connector at the center of the Apple IIc back panel is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms. This arrangement is suitable for most video monitors.

Figure 11-24. Video Output Back Panel Connectors



Video Expansion Output

The back panel of the Apple IIc has a DB-15 connector for sophisticated video interfaces external to the computer. Figure 11-25 shows the pin assignments for this connector; Table 11-17 describes the signals. In Table 11-17, the column labeled *Deriv* indicates what clock signals the video signals are derived from. LDPS, CREF, and PRAS have a maximum delay of 30 ns from the appropriate 14-MHz rising edge. SEROUT is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of ϕ 1.

To align CREF so it is in the same phase at the beginning of every line, certain clock signals must be stretched. This stretch is for one 7M cycle (140 ns), and occurs at the end of each video line. All timing signals except 14M, 7M, and CREF are stretched.

▲Warning

The maximum allowable current drain of +12V regulated power at the video expansion connector is 300 milliamps. If the external device draws more than this, it can damage the computer or cause the power supply to shut down.

▲Warning

The signals at the DB-15 on the Apple IIc are not the same as those at the DB-15 on the Apple III. Do not attempt to plug a cable intended for one into the other.

Several of these signals, such as 14 MHz, must be buffered within about four inches (10 cm) of the back panel connector—preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support.

Figure 11-25. The Video Expansion Connector Pinouts

8	7	6	5	4	3	2	1
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	
•	•	•	•	•	•	•	

Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14M	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

Table 11-17. The Video Expansion Connector Signals

Pin #	Deriv	Name	Description
1	$\phi 0$	TEXT	Video text signal from TMG; set to inverse of GR, except in double-high-resolution mode
2		14M	14-MHz master timing signal from the system oscillator
3	Q3	SYNC*	Displays horizontal and vertical synchronization signal from IOU pin 39
4	PRAS	SEGB	Displays vertical counter bit from IOU pin 4; in text mode indicates second low-order vertical counter; in graphics mode indicates low-resolution

Table 11-17—continued. The Video Expansion Connector Signals

Pin #	Deriv	Name	Description
5		1VSOUND	One-volt sound signal from pin 5 of the audio hybrid circuit (AUD)
6	14M	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WNDW*	Active area display blanking; includes both horizontal and vertical blanking
8		+12V	Regulated +12 volts DC; can drive 300 mA
9	14M	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14M	SEROUT*	Serialized character-generator output from pin 1 of the 74LS166 shift register
12		NTSC	Composite NTSC video signal from VID hybrid chip
13		GND	Ground reference and supply
14	$\phi 0$	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14M	CREF	Color reference signal from TMG pin 3; 3.58 MHz

11.10 Disk I/O

Disk I/O—for both the built-in and the external drive—is supported by the IWM disk controller unit. The external drive is attached via a DB-19 connector. Figure 11-26 shows this connector. Table 11-18 describes the pin assignments. Supply voltages come from the power supply; all other signals come from the IWM, described in Section 11.5.5.

▲ Warning

The power available at this connector is for a Disk IIc or similar drive only. Do not use power from the external disk connector for any other purpose—you may damage the internal voltage converter. To derive external power for an attached device, use one of the other connectors and observe the current limits given in this manual.

Figure 11-26. Disk Drive Connector

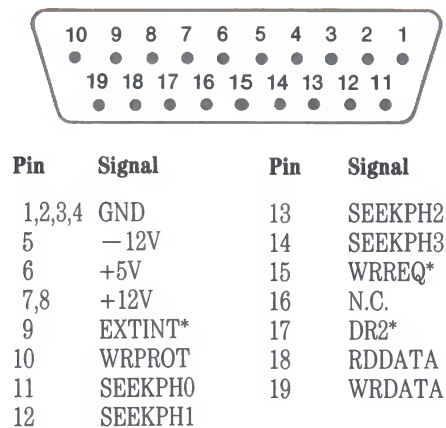


Table 11-18. Disk Drive Connector Signals

Connector		
Pin #	Name	Description
1, 2, 3, 4	GND	Ground reference and supply
6	+5V	+5 volt supply
7, 8	+12	+12 volt supply
9	EXTINT*	External interrupt
10	WRPROT	Write-protect input
11-14	ϕ 0-4	Motor phase 0-4 output
15	WRREQ*	Write request
17	DR1*	Drive 1 select
18	RDDATA	Read data input
19	WRDATA	Write data output

11.11 Serial I/O

The Apple IIc has built into it two 6551 asynchronous communication interface adapters (ACIA) and supporting input and output buffers for full-duplex serial communication. Figure 11-27 is a block diagram of the Apple IIc serial ports. ACIA outputs are buffered by a 1448-quad line driver. Similarly, ACIA inputs are buffered by a 1489-quad line receiver.

Figure 11-27. Serial Port Circuits

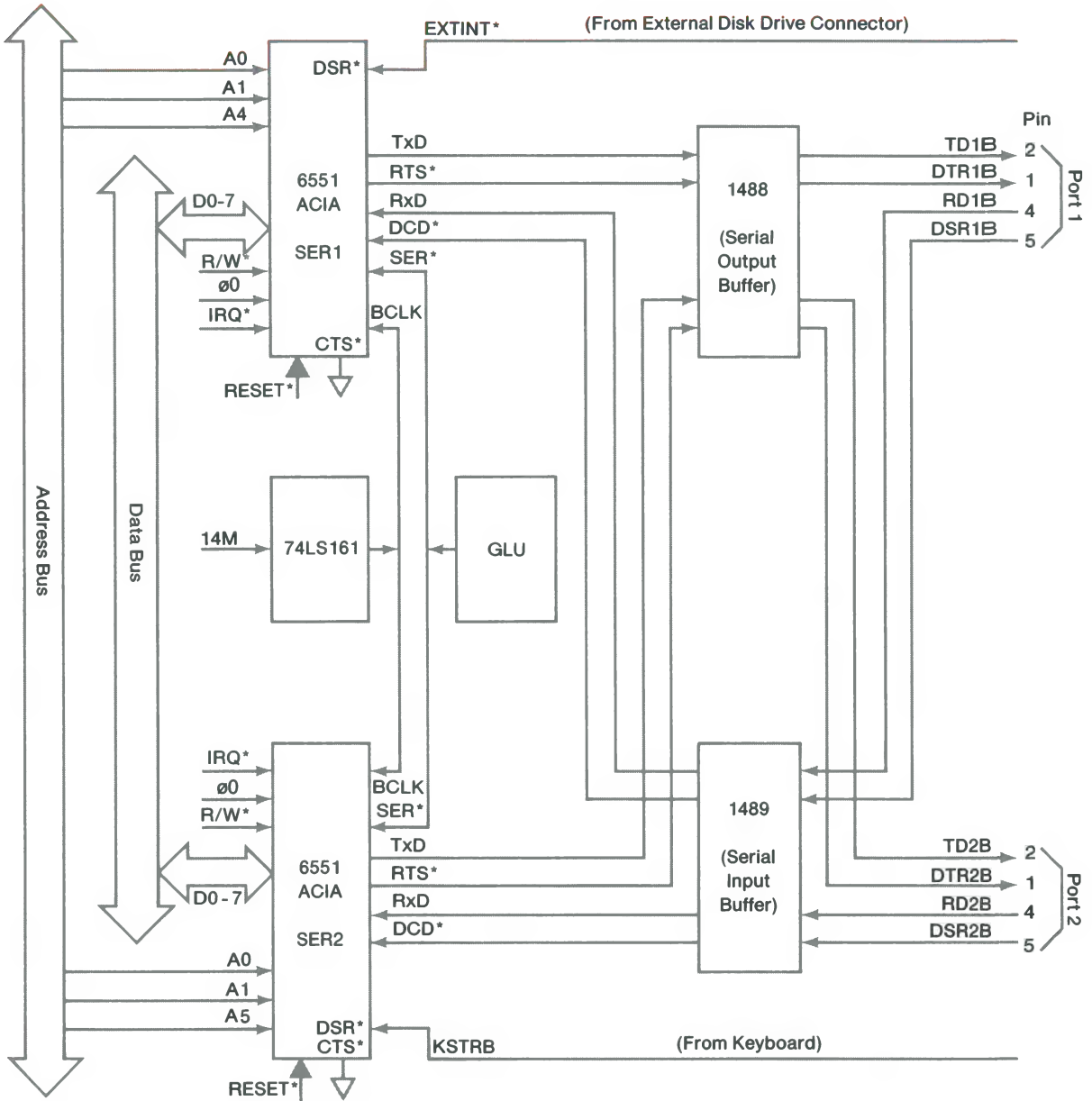
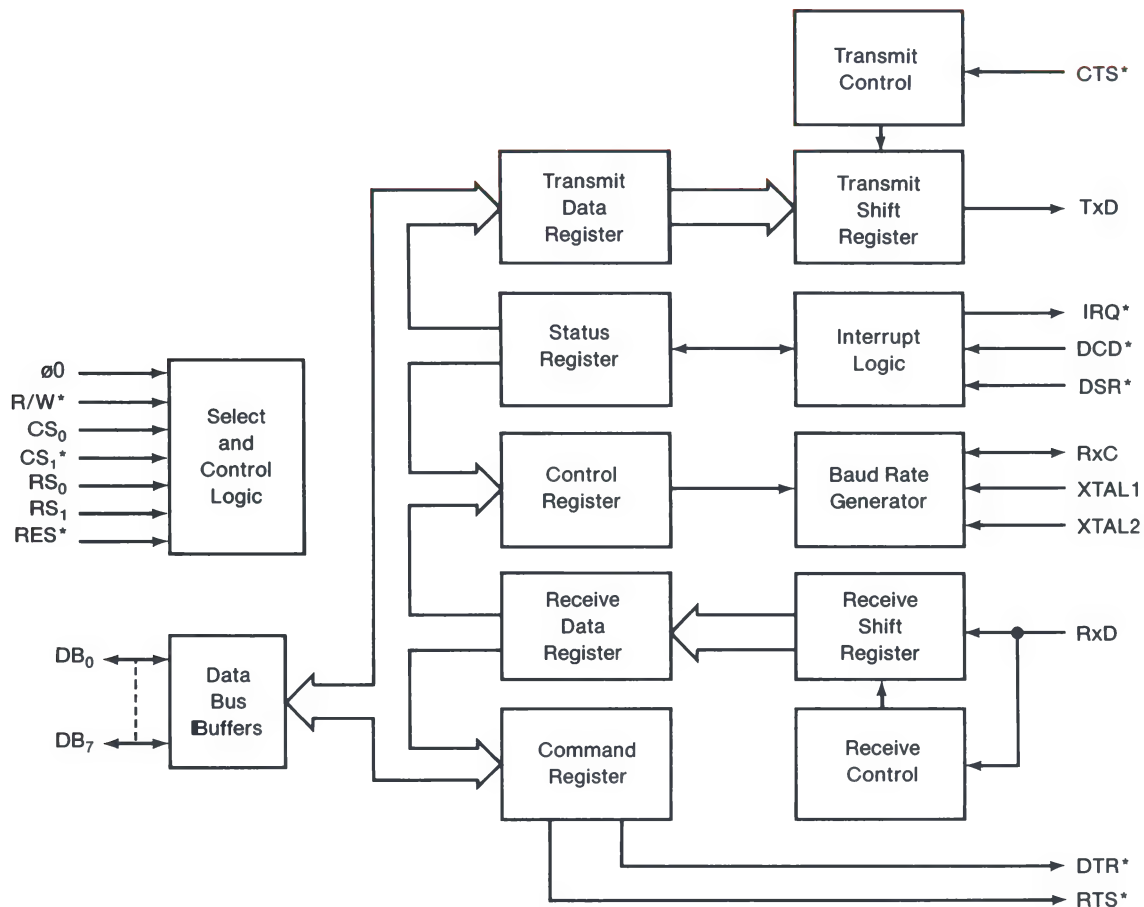


Figure 11-28 is a detailed block diagram of the 6551 ACIA. The registers are described in Sections 11.11.1 through 11.11.4.

Figure 11-28. 6551 ACIA Block Diagram

Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



The 6551 pin assignments are shown in Figure 11-29 and described in Table 11-19. Note that the two 6551s are not used in exactly the same way—each one supports a different set of interrupts.

Port 1 reads external interrupts ($EXTINT^*$) on its Data Set Ready (DSR) pin. This input is tied to +5V through a 3.3-K Ω pullup resistor.

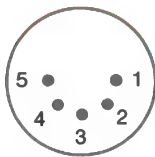
Figure 11-29. The 6551 Pinouts

GND	1	28	R/W*
A5	2	27	$\phi 0$
SER*	3	26	IRQ*
RESET*	4	25	D7
(N.C.)	5	24	D6
BCLK	6	23	D5
(N.C.)	7	22	D4
RTS*	8	21	D3
CTS*	9	20	D2
TxD	10	19	D1
(N.C.)	11	18	D0
RxD	12	17	DSR*
A0	13	16	DCD*
A1	14	15	+5V

Table 11-19. The 6551 Signal Descriptions

Pin #	Name	Description
1	GND	Power and signal common ground
2	A4 A5	Address line 4 to select serial port 1 Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	N.C.	Not connected
6	BCLK	Baud rate clock from GLU
7	N.C.	Not connected
8	RTS*	Request to Send output
9	CTS*	Clear to Send input (not used on IIC; tied to g round)
10	TXD	Transmit Data output
11	N.C.	Not connected
12	RXD	Receive Data input
13, 14	A0, A1	Address lines 0 and 1
15	+5V	+5 volt supply
16	DSR	DCD* pin; used on IIC as Data Set Ready input
17	EXTINT* KSTRB	DSR* pin; used on IIC as External interrupt (port 1 ACIA), or Keyboard strobe input (port 2 ACIA; Appendix E)
18-25	D0-D7	8-bit data bus
26	IRQ*	Interrupt Request input
27	$\phi 0$	Phase 0 clock pulse
28	R/W*	Read/write select input

Figure 11-30. Serial Port Connectors



Pin	Port 1	Port 2
1	DTR1B	DTR2B
2	TD1B	TD2B
3	GND	GND
4	RD1B	RD2B
5	DSR1B	DSR2B

Table 11-20. Serial Port Connector Signals

Pin #	Name	Description
1	DTR1B DTR2B	Data Terminal Ready output
2	TD1B TD2B	Transmit Data output
3	GND	Power and signal common
4	RD1B RD2B	Read Data input
5	DSR1B DSR2B	Data Set Ready input

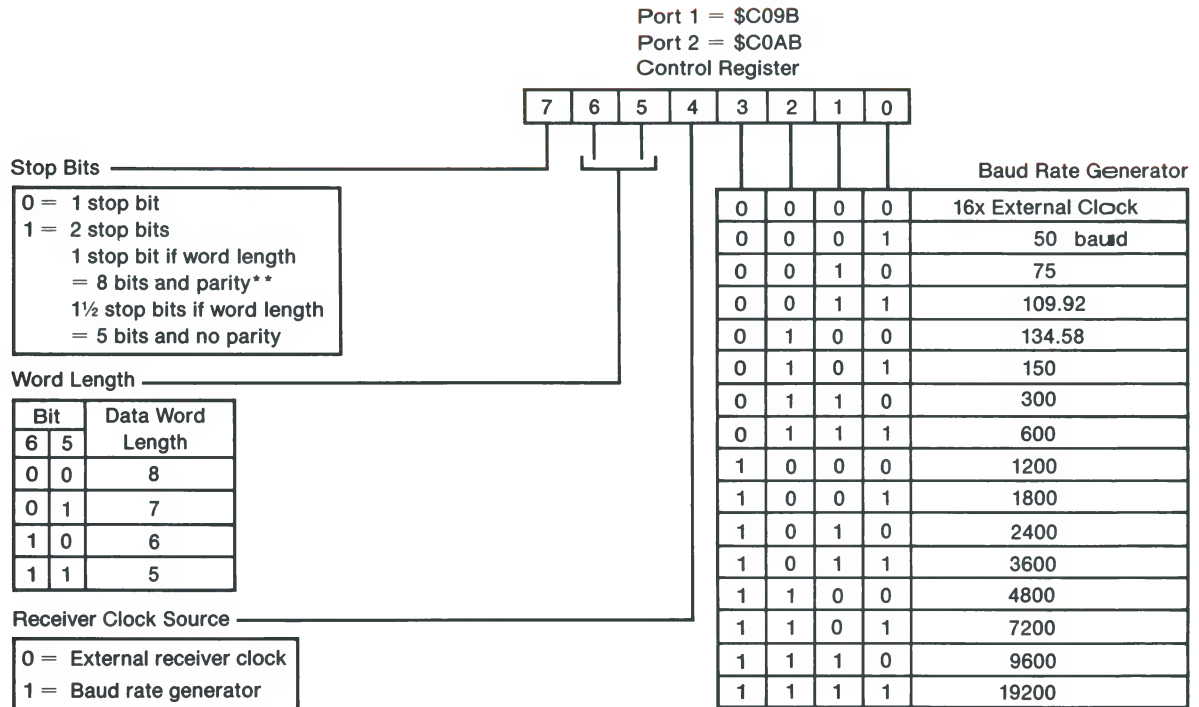
11.11.1 ACIA Control Register

Figure 11-31 shows the bit assignments for the ACIA control register, which the hardware locates at address \$C09B for serial port 1, and \$C0AB for serial port 2. This register determines the number of data and stop bits the ACIA will transmit and receive, and the clock source and baud rate to use for data transfer.

The receiver clock source is derived from the Apple IIc's TMG chip; the resulting baud rates are equal to or up to 2 percent lower than the nominal rate. (The EIA standard allows plus or minus 2 percent variation.) If an Apple IIc serial port is used with a modem that is 2 percent above the nominal rate, framing errors can occur, especially at 1200 baud and above, when using eight data bits. It may be necessary to select a lower baud rate for 8-bit binary data transfers.

Figure 11-31. ACIA Control Register

Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



**This allows for 9-bit transmission (8 data plus parity).

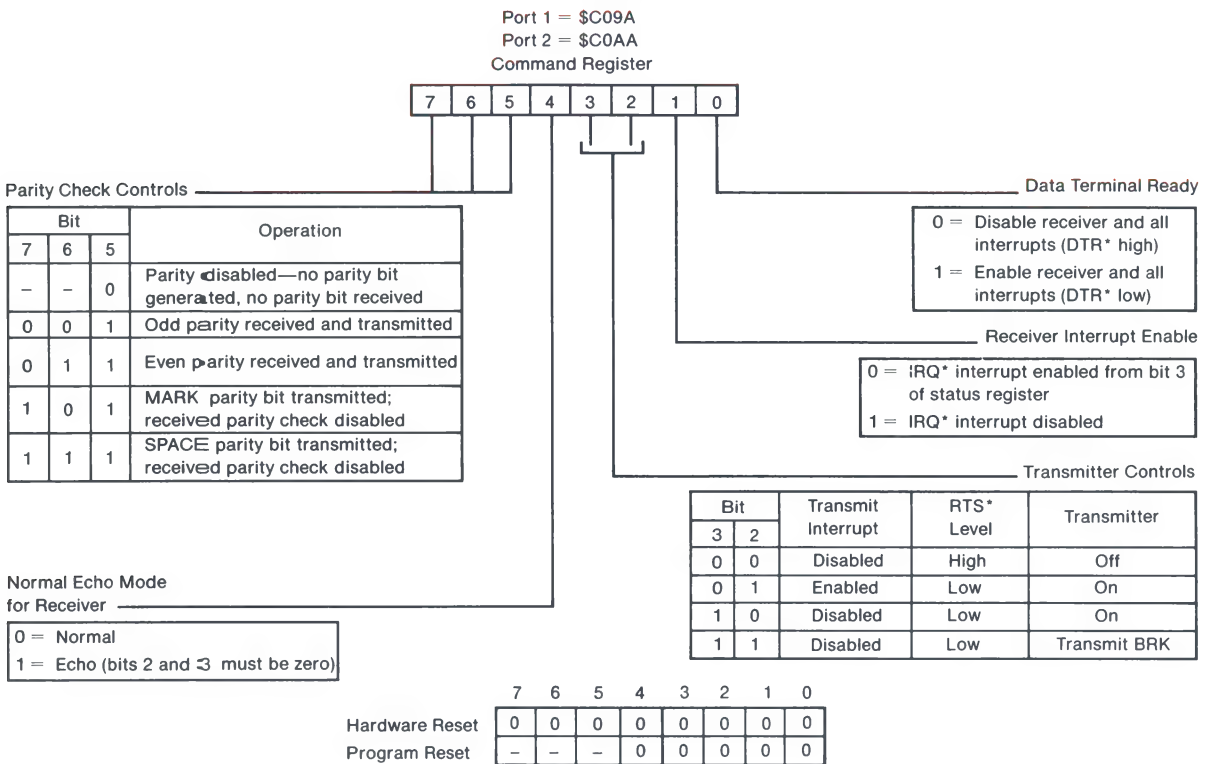
	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	—	—	—	—	—	—	—	—

11.11.2 ACIA Command Register

Figure 11-32 shows the bit assignments for the ACIA command register, which the hardware locates at address \$C09A for serial port 1, and at \$C0AA for serial port 2. This register controls specific transmit and receive functions: parity checking, echoing input to output, allowing transmit and receive interrupts, and setting levels for Data Terminal Ready and Request to Send.

Figure 11-32. ACIA Command Register

Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.

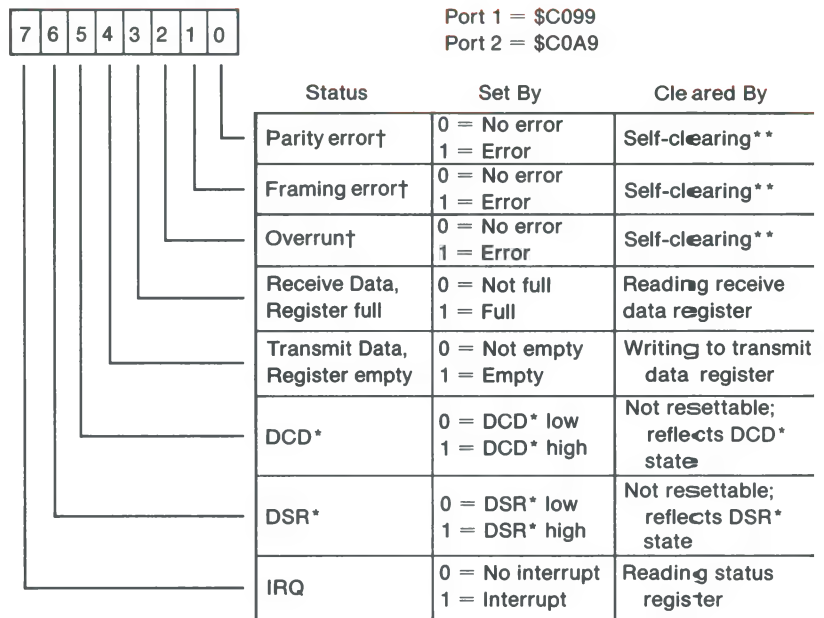


11.11.3 ACIA Status Register

Figure 11-33 shows the bit assignments for the ACIA status register, which is hard-wired to address \$C099 for serial port 1, and \$C0A9 for serial port 2. This register reports the condition of the transmit/receive register, errors detected during data transfer, and the level of the Data Carrier Detect, Data Set Ready, and Interrupt Request lines.

Figure 11-33. ACIA Status Register

Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 300 1 Stender, Santa Clara, CA 95052.



† No interrupt generated for these conditions.

** Cleared automatically after a read of RDR and the next error-free receipt of data.

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	-	-	-	-	-	-	-	-

11.11.4 ACIA Transmit/Receive Register

Each ACIA uses the same address—\$C098 for serial port 1, \$C0A8 for serial port 2—as temporary storage for both transmission and reception of data.

When the register is used for transmitting data, bit 0 is the leading bit to be transmitted; unused data bits are the high-order bits, which are ignored.

When the register is used for receiving data, bit 0 is the first bit received; unused data bits are the high-order bits, which are set to 0. Parity bits never appear in the receive data register; they are stripped off after being used for external parity checking.

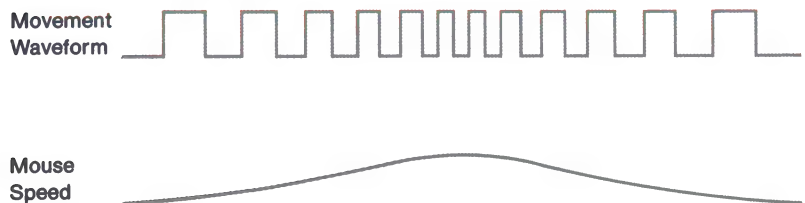
11.12 Mouse Input

The mouse is a hand-held X-Y pointing device that can be rolled along a flat surface. It has an attached pushbutton. This section describes how mouse movement and direction can be detected and interpreted.

A mouse has a ball inside its housing that protrudes a small distance so that its turning corresponds to mouse movements across a table top. Two wheels inside the housing, set at 90-degree angles to each other, follow movements of the ball; this causes two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble circular slide mounts used with stereoscopic slide viewers.

The light from a tiny infrared emitter reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse causes the resulting voltage to swing quickly to a 1 or a 0 value as soon as a certain threshold is crossed. The result is something approximating a square wave (Figure 11-34) that varies directly with the speed of mouse movement. One of these indicates the X component (X0) of mouse movement; the other, the Y component (Y0).

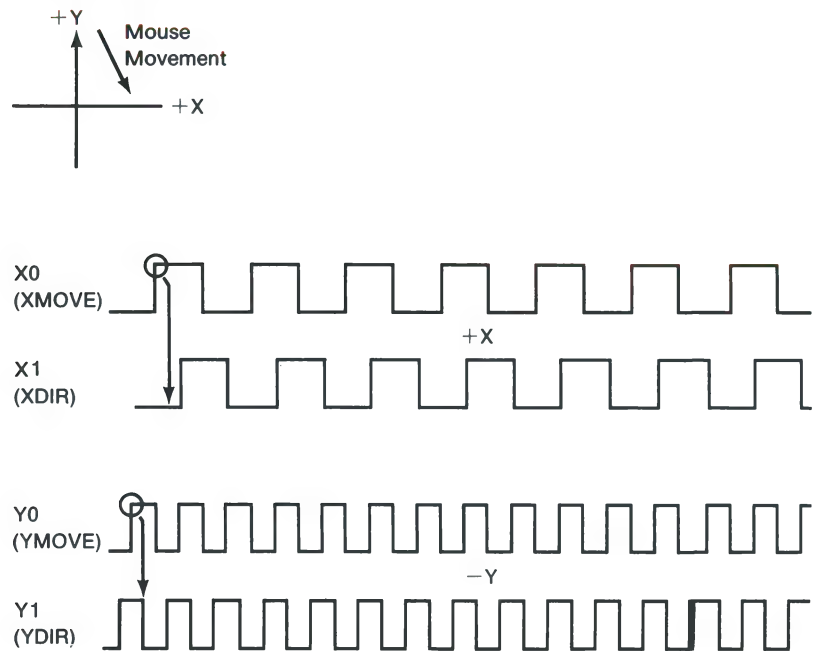
Figure 11-34. Sample Mouse Waveform



Under program control, either the rising edge or the falling edge of each square wave can cause an interrupt, which the firmware handles by updating a counter. However, the program needs to know whether to add or to subtract 1 from a counter; that is, it needs to know the direction of X or Y movement.

There is a second infrared emitter/photoreceptor pair almost 180 degrees opposite the first pair for each disk. These pairs are positioned in such a way that the square waves they generate are approximately a quarter-wave offset from their respective movement waves (Figure 11-35). These waveforms are called X1 (X direction) and Y1 (Y direction).

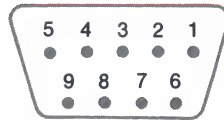
Figure 11-35. Mouse Movement and Direction Waveforms



When a rising edge of X0 causes an interrupt, a mouse-driver program can immediately check whether X1 is 0 (indicating a movement to the right) or 1 (indicating a movement to the left). Similarly, the mouse driver can read Y1 immediately after a Y0 interrupt to determine whether the mouse moved up or down one count along the Y axis.

Figure 11-36 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-21 gives the signal names and descriptions.

Figure 11-36. Mouse Connector



Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

Table 11-21. Mouse Connector Signals

Pin #	Name	Description
1	MOUSEID*	Mouse identifier: when active, disables NE556 hand controller timer
2	+5V	Total current drain from this pin must not exceed 100 mA.
3	GND	System ground
4	XDIR	Mouse X-direction indicator
5	XMOVE	Mouse X-movement interrupt
6	N.C.	Not connected
7	MSW*	Mouse button
8	YDIR	Mouse Y-direction indicator
9	YMOVE	Mouse Y-movement interrupt

Figure 11-37 shows the mouse and hand controller circuitry with the mouse circuits emphasized. Figure 11-38 illustrates the values of the mouse-button circuit when the button is pressed or not pressed. Pressing the button disables the NE556 by pulling the reset comparator threshold value up so that it cannot reset the flip flop. As a result the mouse-button input value remains at a TTL level.

Figure 11-37. Mouse Circuits

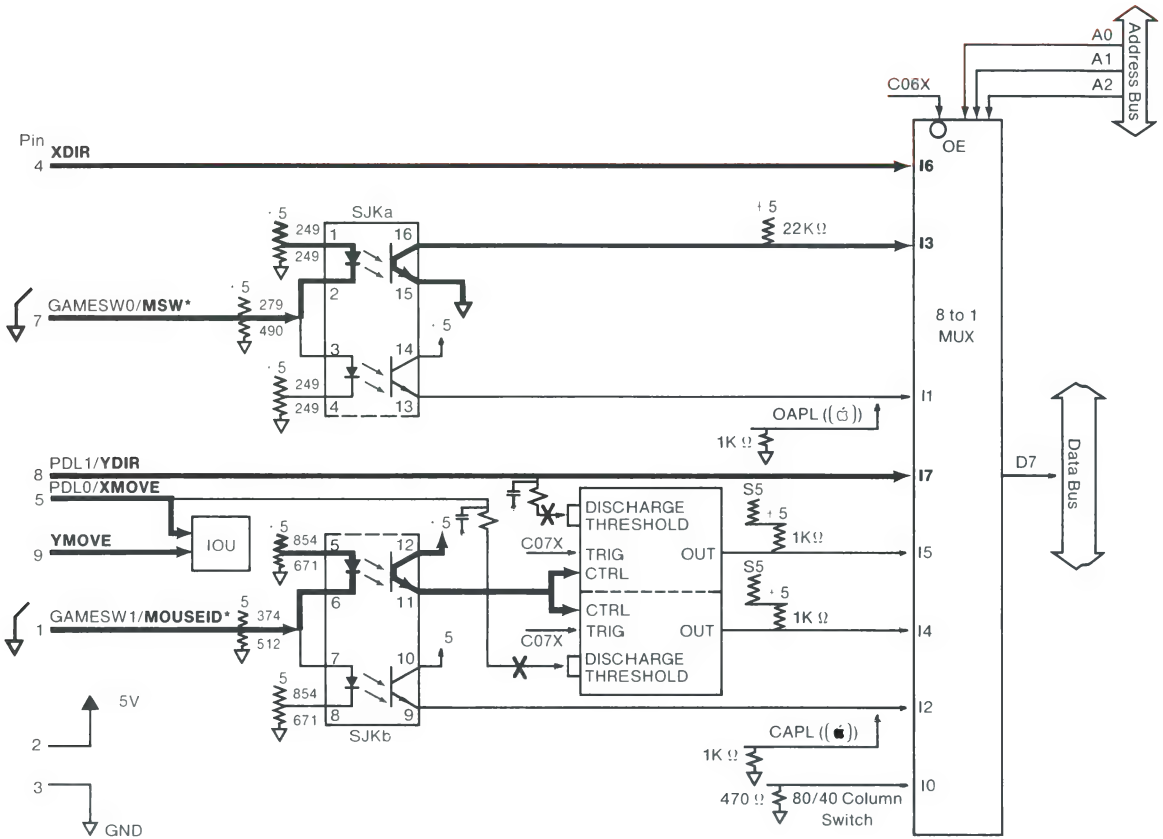
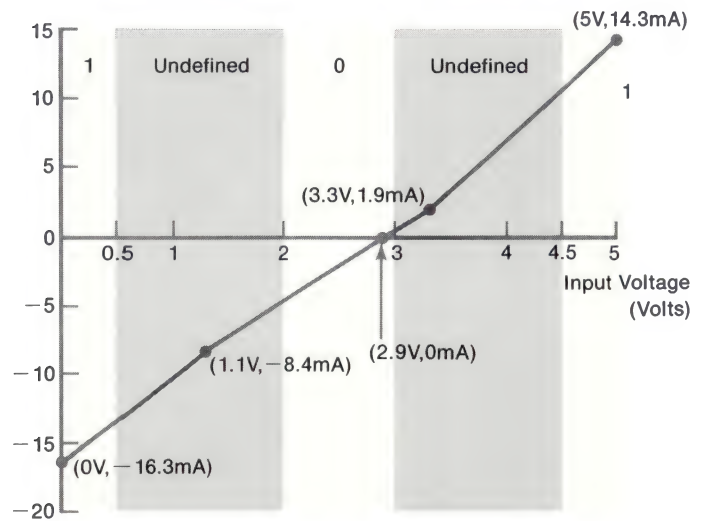
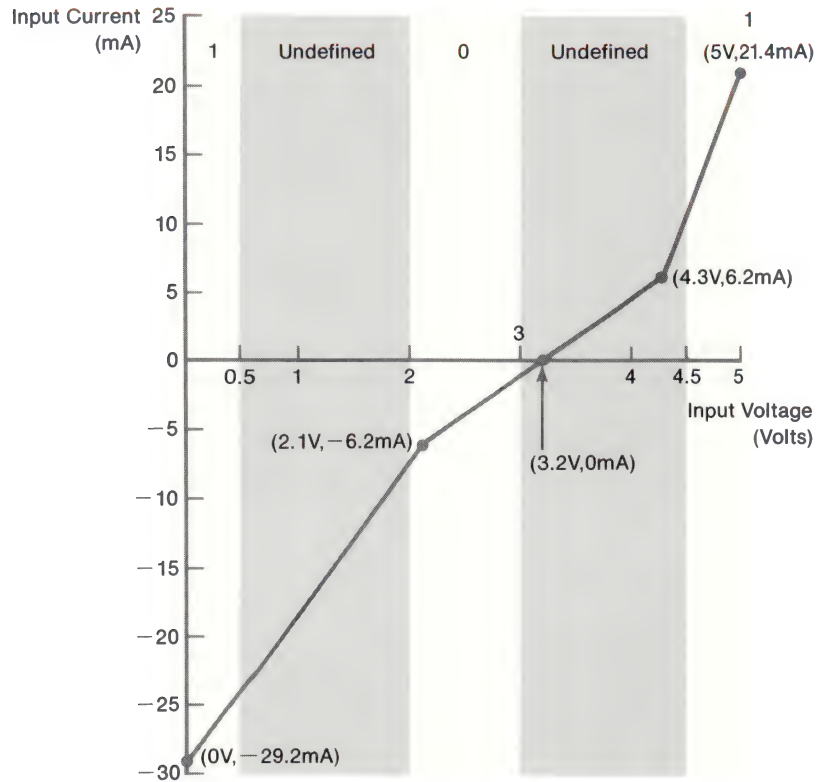


Figure 11-38. Mouse Button Signals



11.13 Hand Controller Input

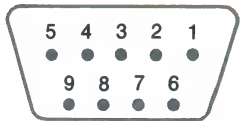
Several input signals that are individually controlled via soft switches are collectively referred to as the hand controller (game) signals. These signals arrive in the Apple IIc via the same DB-9 connector as the one used for the mouse (Section 11.12), but the Apple IIc interprets these signals differently.

The DB-9 connector pin assignments and signal descriptions, as used for hand controller input, appear in Figure 11-39 and Table 11-22.

Even though they are normally used for hand controllers, these signals can be used for other simple I/O applications. There are two 1-bit switch inputs, labeled Sw0 and Sw1, and two analog inputs, called paddles and labeled Pdl0 and Pdl1. Figure 11-40 shows how to connect the 1-bit switch inputs for compatibility with all other Apple II series computers.

The switch inputs are multiplexed by a 74LS251 8-to-1 multiplexer enabled by the C06X* signal from the MMU. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus. Figure 11-41 shows the mouse and hand controller circuitry with the hand controller circuits highlighted. Figure 11-42 illustrates the values of the hand controller switch inputs when the switch is open or closed.

Figure 11-39. Hand Controller Connector



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controllers
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controllers

Table 11-22. Hand Controller Connector Signals

Pin #	Name	Description
1	GAMESW1	Switch input 1 (sometimes called paddle button 1)
2	+5V	+5V power supply; total current drain from this pin must not exceed 100 mA.
3	GND	System ground
4, 9		Not used for hand controllers
5, 8	PDL0 and PDL1	Hand controller inputs; each of these must be connected to a 150-K Ω variable resistor connected to +5V.
6	N.C.	Not connected
7	GAMESW0	Switch input 0 (sometimes called paddle button 0)

Figure 11-40. How to Connect Switch Inputs

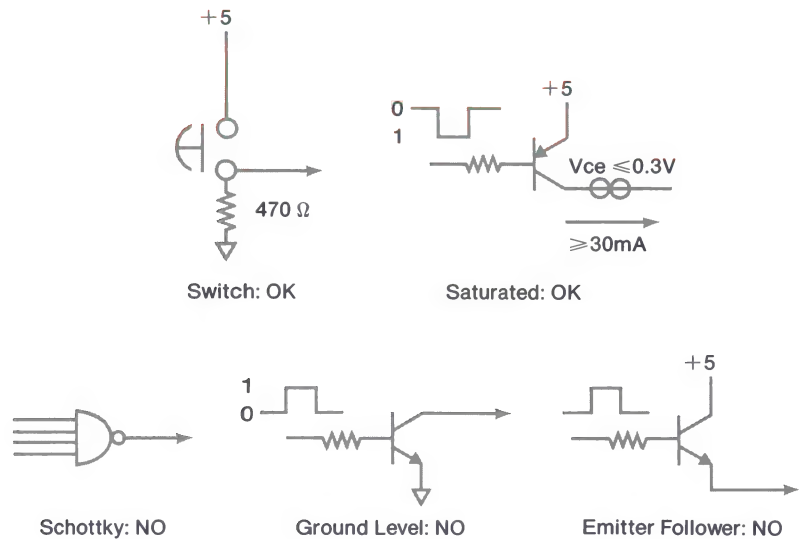


Figure 11-41. Hand Controller Circuits

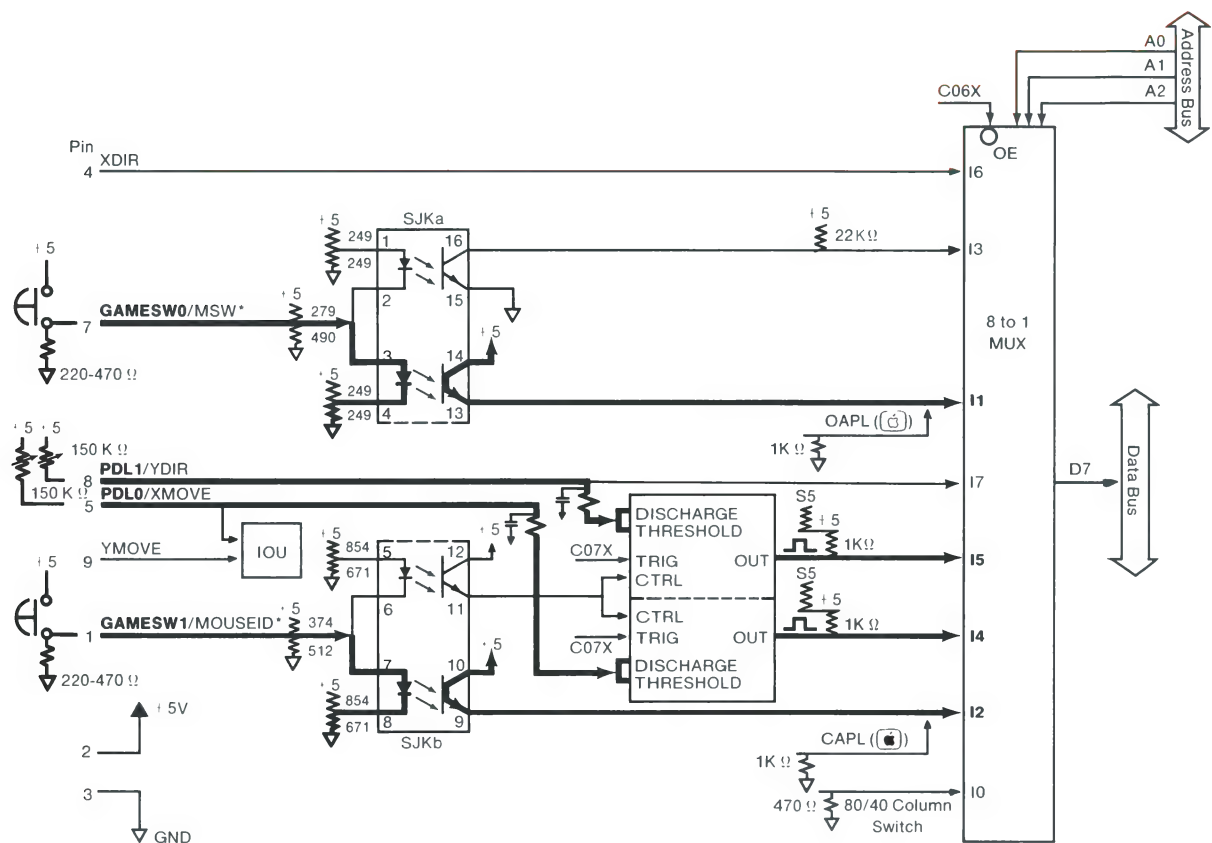
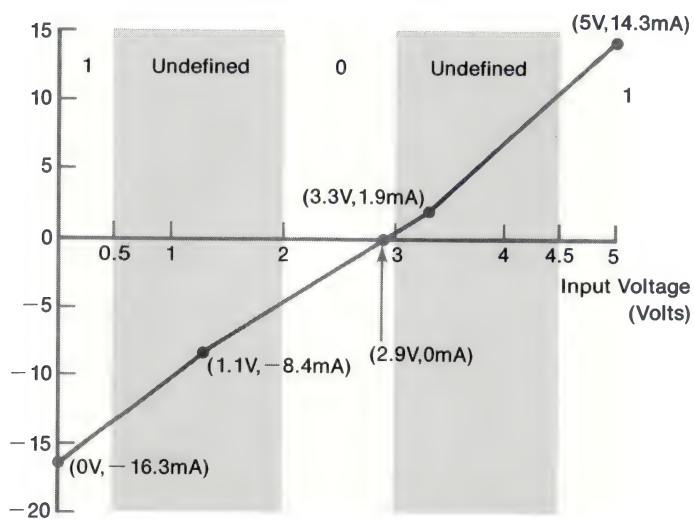
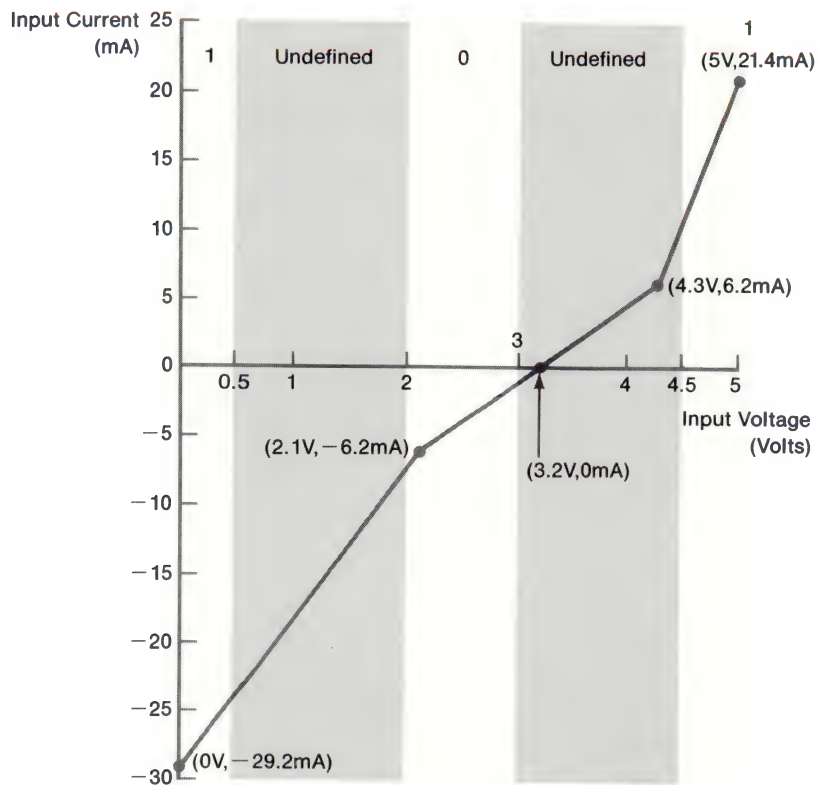


Figure 11-42. Hand Controller Signals



The hand controller inputs are connected to the timing inputs of an NE556 dual analog timer. Addressing \$C07X sends a signal from MMU pin 22 that resets both timers and causes their outputs to go to 1 (high). A variable resistance of up to 150 K Ω connected between one of these inputs and the +5V supply controls the charging time of one of the two 0.022 microfarad capacitors.

When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

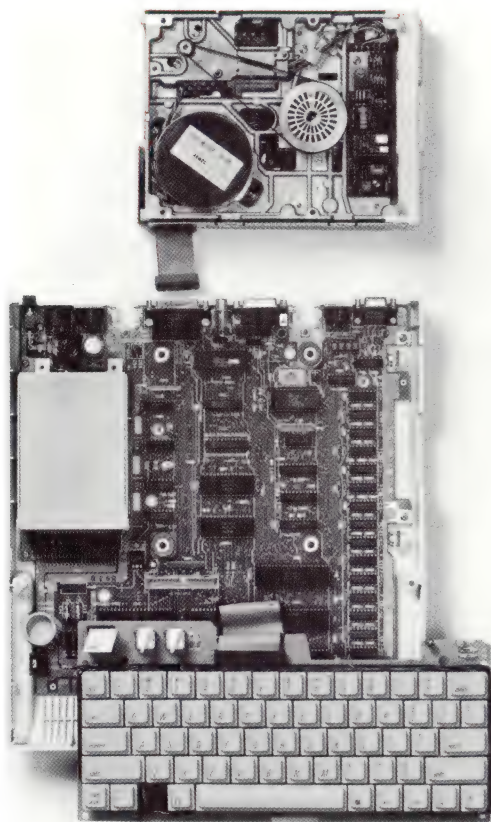
▲Warning

The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. If you retrigger or read the other paddle too soon (that is, in less than 3 ms), you will get a false value for it.

11.14 Schematic Diagrams

The following pages contain schematic diagrams for the Apple IIc.





This appendix describes the differences between the 6502 and the 65C02 microprocessors. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in numbers of cycles. One cycle for the Apple IIc equals 0.978 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of 65C02 instructions present on the 6502.

A.1 Differences Between 6502 and 65C02

The data sheet in Section A.2 lists the new 65C02 instructions and addressing modes. This section supplements that information by listing the instructions whose execution times or results have changed from their 6502 counterparts.

A.1.1 Differing Cycle Times

In general, differences in execution times are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed execution times are few.

Table A-1 lists the 65C02 instructions whose number of instruction execution cycles is different from their number on the 6502.

Table A-1. Cycle Time Differences

Instruction/Mode	Opcode	6502 Cycles	65C02 Cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

A.1.2 Differing Instruction Results

The instructions that have different results from their 6502 equivalents are

- ☐ BIT (in immediate mode)
- ☐ JMP (indirect, when crossing a page boundary).

The BIT instruction when used in immediate mode (code \$89) leaves processor status register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the status register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location being tested contains a 0.

If the JMP indirect instruction (code \$6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP (\$02FF) gets ADL from location \$02FF on both processors. On the 65C02, ADH comes from \$0300 while on the 6502, ADH comes from \$0200.

A.2 Data Sheet

The rest of this appendix is copyright 1982, NCR Corporation, Dayton, Ohio, and is reprinted with their permission.

■ GENERAL DESCRIPTION

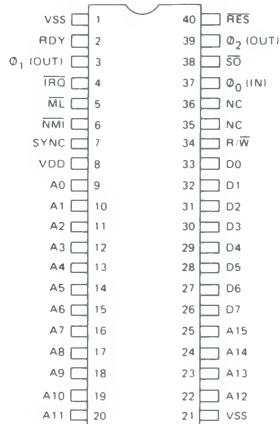
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

■ FEATURES

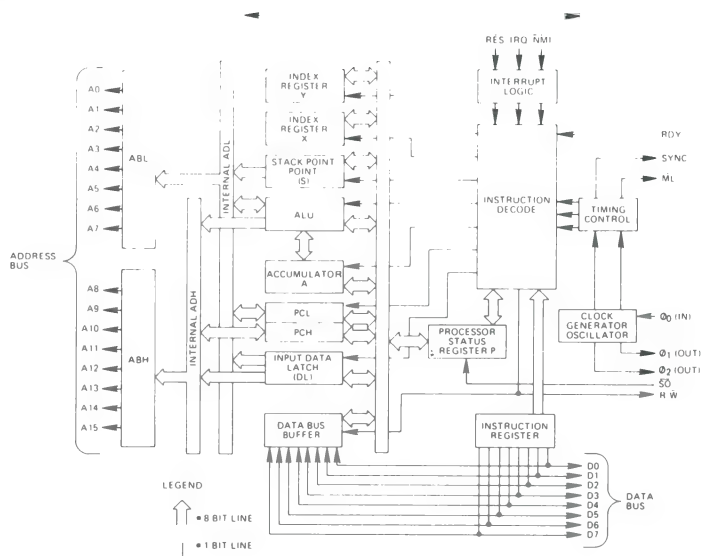
- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 Hz for even lower power consumption (pseudo-static: stop during ϕ_2 high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ, $\overline{S\overline{O}}$, NMI and RES)

* Specifications are subject to change without notice.

■ PIN CONFIGURATION



■ NCR65C02 BLOCK DIAGRAM



NCR65C02**■ ABSOLUTE MAXIMUM RATINGS:** ($V_{DD} = 5.0\text{ V} \pm 5\%$, $V_{SS} = 0\text{ V}$, $T_A = 0^\circ\text{ to } +70^\circ\text{C}$)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	-0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	-0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to +70	$^\circ\text{C}$
STORAGE TEMP.	T_{STG}	-55 to +150	$^\circ\text{C}$

■ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{\text{IRQ}}^*$	Interrupt Request
RDY^*	Ready
$\overline{\text{ML}}$	Memory Lock
$\overline{\text{NMI}}^*$	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{\text{RES}}^*$	Reset
SO^*	Set Overflow
NC	No Connection
R/ $\overline{\text{W}}$	Read/Write
V_{DD}	Power Supply (+5V)
V_{SS}	Internal Logic Ground
ϕ_0	Clock Input
ϕ_1, ϕ_2	Clock Output

*This pin has an optional internal pullup for a No Connect condition.

■ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage ϕ_0 (IN)	V_{IH}	$V_{SS} + 2.4$	—	V_{DD}	V
Input High Voltage $\overline{\text{RES}}, \overline{\text{NMI}}, \text{RDY}, \overline{\text{IRQ}}, \text{Data}, \text{S.O.}$		$V_{SS} + 2.0$	—	—	V
Input Low Voltage ϕ_0 (IN)	V_{IL}	$V_{SS} - 0.3$	—	$V_{SS} + 0.4$	V
$\overline{\text{RES}}, \overline{\text{NMI}}, \text{RDY}, \overline{\text{IRQ}}, \text{Data}, \text{S.O.}$		—	—	$V_{SS} + 0.8$	V
Input Leakage Current ($V_{IN} = 0$ to 5.25V, $V_{DD} = 5.25\text{V}$)	I_{IN}	—	—	—	μA
With pullups		-30	—	+30	μA
Without pullups		—	—	+1.0	μA
Three State (Off State) Input Current ($V_{IN} = 0.4$ to 2.4V, $V_{CC} = 5.25\text{V}$)		—	—	—	μA
Data Lines	I_{TSI}	—	—	10	μA
Output High Voltage ($I_{OH} = -100\ \mu\text{A}$, $V_{DD} = 4.75\text{V}$)	V_{OH}	$V_{SS} + 2.4$	—	—	V
$\text{SYNC}, \text{Data}, \text{A0-A15}, \text{R/W}$					
Out Low Voltage ($I_{OL} = 1.6\text{mA}$, $V_{DD} = 4.75\text{V}$)	V_{OL}	—	—	$V_{SS} + 0.4$	V
$\text{SYNC}, \text{Data}, \text{A0-A15}, \text{R/W}$					
Supply Current $f = 1\text{MHz}$	I_{DD}	—	—	4	mA
Supply Current $f = 2\text{MHz}$	I_{DD}	—	—	8	mA
Capacitance ($V_{IN} = 0$, $T_A = 25^\circ\text{C}$, $f = 1\text{MHz}$)	C	—	—	—	pF
Logic	C_{IN}	—	—	5	
Data		—	—	10	
A0-A15, R/W, SYNC	C_{out}	—	—	10	
ϕ_0 (IN)	C_{ϕ_0} (IN)	—	—	10	

■ AC CHARACTERISTICS $V_{DD} = 5.0V \pm 5\%$, $T_A = 0^\circ C$ to $70^\circ C$, Load = 1 TTL + 130 pF

		1MHz		2MHz		3MHz		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Unit
Delay Time, θ_0 (IN) to θ_2 (OUT)	t_{DLY}	—	60	—	60	20	60	nS
Delay Time, θ_1 (OUT) to θ_2 (OUT)	t_{DLY1}	—20	20	—20	20	—20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	μS
Clock Pulse Width Low	t_{PL}	460	—	220	—	160	—	nS
Clock Pulse Width High	t_{PH}	460	—	220	—	160	—	nS
Fall Time, Rise Time	t_F, t_R	—	25	—	25	—	25	nS
Address Hold Time	t_{AH}	20	—	20	—	0	—	nS
Address Setup Time	t_{ADS}	—	225	—	140	—	110	nS
Access Time	t_{ACC}	650	—	310	—	170	—	nS
Read Data Hold Time	t_{DHR}	10	—	10	—	10	—	nS
Read Data Setup Time	t_{DSU}	100	—	60	—	60	—	nS
Write Data Delay Time	t_{MDS}	—	30	—	30	—	30	nS
Write Data Hold Time	t_{DHW}	20	—	20	—	15	—	nS
\overline{SO} Setup Time	t_{SO}	100	—	100	—	100	—	nS
Processor Control Setup Time**	t_{PCS}	200	—	150	—	150	—	nS
SYNC Setup Time	t_{SYNC}	—	225	—	140	—	100	nS
ML Setup Time	t_{ML}	—	225	—	140	—	100	nS
Input Clock Rise/Fall Time	t_{F00}, t_{R00}	—	25	—	25	—	25	nS

*NCR65C02 can be held static with θ_2 high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

■ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

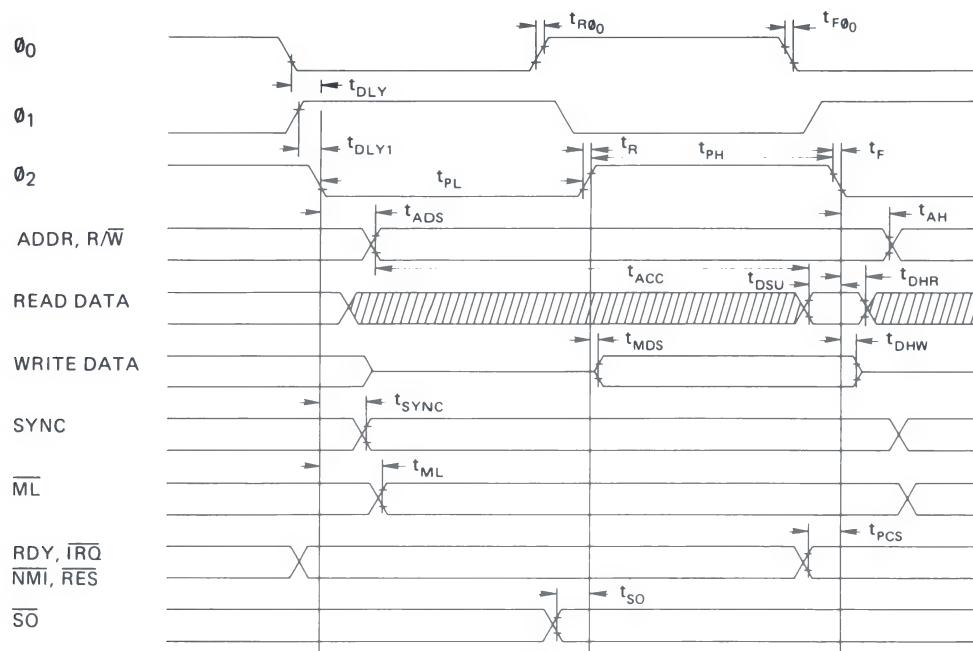
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor																					
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.																					
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). <table> <tr> <td>Op Code</td><td>Bytes</td><td>Cycles</td></tr> <tr> <td>X2</td><td>2</td><td>2</td></tr> <tr> <td>X3, X7, XB, XF</td><td>1</td><td>1</td></tr> <tr> <td>44</td><td>2</td><td>3</td></tr> <tr> <td>54, D4, F4</td><td>2</td><td>4</td></tr> <tr> <td>5C</td><td>3</td><td>8</td></tr> <tr> <td>DC, FC</td><td>3</td><td>4</td></tr> </table>	Op Code	Bytes	Cycles	X2	2	2	X3, X7, XB, XF	1	1	44	2	3	54, D4, F4	2	4	5C	3	8	DC, FC	3	4
Op Code	Bytes	Cycles																					
X2	2	2																					
X3, X7, XB, XF	1	1																					
44	2	3																					
54, D4, F4	2	4																					
5C	3	8																					
DC, FC	3	4																					
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.																					
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.																					
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.																					
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.																					
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.																					

■ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during θ_2 .
Unused input-only pins (\overline{IRQ} , \overline{NMI} , RDY, RES, \overline{SO}).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

■ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

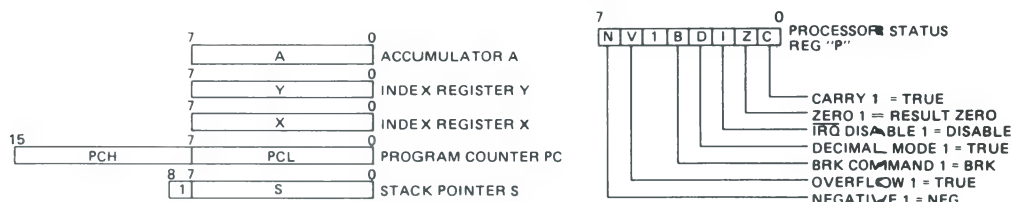
■ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG, X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

■ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND, X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

■ MICROPROCESSOR PROGRAMMING MODEL



■ FUNCTIONAL DESCRIPTION

Timing Control

The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

Program Counter

The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

Instruction Register and Decode

Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

Accumulator

The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

Index Registers

There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

Stack Pointer

The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts (NMI and IRQ). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

Processor Status Register

The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

NCR65C02

■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

Implied Addressing [Implied]

In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]

This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]

With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

Absolute Addressing [Absolute]

For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]

Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]

Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]

Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]

Relative addressing is used only with branch instructions;

it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is -128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]

With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**

With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]

This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**

In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)

The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

■ SIGNAL DESCRIPTION

Address Bus (A0-A15)

A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks (Φ_0 , Φ_1 , and Φ_2)

Φ_0 is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The Φ_2 clock output is in phase with Φ_0 . The Φ_1 output pin is 180° out of phase with Φ_0 . (See timing diagram.)

Data Bus (D0-D7)

The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{\text{IRQ}}$)

This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{\text{IRQ}}$ is sampled during Φ_2 operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during Φ_1 . The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{\text{IRQ}}$ s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{\text{ML}}$)

In a multiprocessor system, the $\overline{\text{ML}}$ output indicates the need to defer the arbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{\text{ML}}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{\text{NMI}}$)

A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{\text{NMI}}$ is sampled during Φ_2 ; the current instruction is completed and the interrupt sequence begins during Φ_1 . The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{\text{NMI}}$ can occur before the first is finished. Care should be taken when using $\overline{\text{NMI}}$ to avoid this.

Ready (RDY)

This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one (Φ_1), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (Φ_2) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{\text{RES}}$)

This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after VDD reaches operating voltage from a power down. A positive transition on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{\text{RES}}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write ($\overline{\text{R/W}}$)

This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{\text{SO}}$)

A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of Φ_1 .

Synchronize (SYNC)

This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during Φ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the Φ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

NCR65C02

■ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC	Add Memory to Accumulator with Carry	LDX	Load Index X with Memory
AND	"AND" Memory with Accumulator	LDY	Load Index Y with Memory
ASL	Shift One Bit Left	LSR	Shift One Bit Right
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set	ORA	"OR" Memory with Accumulator
BEQ	Branch on Result Zero	PHA	Push Accumulator on Stack
BIT	Test Memory Bits with Accumulator	PHP	Push Processor Status on Stack
BMI	Branch on Result Minus	*PHX	Push Index X on Stack
BNE	Branch on Result not Zero	*PHY	Push Index Y on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
*BRA	Branch Always	PLP	Pull Processor Status from Stack
BRK	Force Break	*PLX	Pull Index X from Stack
BVC	Branch on Overflow Clear	*PLY	Pull Index Y from Stack
BVS	Branch on Overflow Set	ROL	Rotate One Bit Left
CLC	Clear Carry Flag	ROR	Rotate One Bit Right
CLD	Clear Decimal Mode	RTI	Return from Interrupt
CLI	Clear Interrupt Disable Bit	RTS	Return from Subroutine
CLV	Clear Overflow Flag	SBC	Subtract Memory from Accumulator with Borrow
CMP	Compare Memory and Accumulator	SEC	Set Carry Flag
CPX	Compare Memory and Index X	SED	Set Decimal Mode
CPY	Compare Memory and Index Y	SEI	Set Interrupt Disable Bit
*DEA	Decrement Accumulator	STA	Store Accumulator in Memory
DEC	Decrement by One	STX	Store Index X in Memory
DEX	Decrement Index X by One	STY	Store Index Y in Memory
DEY	Decrement Index Y by One	*STZ	Store Zero in Memory
EOR	"Exclusive-or" Memory with Accumulator	TAX	Transfer Accumulator to Index X
*INA	Increment Accumulator	TAY	Transfer Accumulator to Index Y
INC	Increment by One	*TRB	Test and Reset Memory Bits with Accumulator
INX	Increment Index X by One	*TSB	Test and Set Memory Bits with Accumulator
INY	Increment Index Y by One	TSX	Transfer Stack Pointer to Index X
JMP	Jump to New Location	TXA	Transfer Index X to Accumulator
JSR	Jump to New Location Saving Return Address	TXS	Transfer Index X to Stack Pointer
LDA	Load Accumulator with Memory	TYA	Transfer Index Y to Accumulator

Note: * = New Instruction

■ MICROPROCESSOR OP CODE TABLE

S	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X				TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL	ORA ind, Y	ORA*† (zpg)			TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs, X	ORA abs, X	ASL abs, X		1
2	JSR	AND abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI	AND rel	AND ind, Y	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg			PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC	EOR rel	EOR ind, Y	EOR*† (zpg)		EOR zpg, X	LSR zpg, X			CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X				STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS	ADC rel	ADC ind, Y	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X		7
8	BRA*	STA rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC	STA rel	STA ind, Y	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY	LDA imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS	LDA rel	LDA ind, Y	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY	CMP imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP zpg	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE	CMP rel	CMP ind, Y	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X		D
E	CPX	SBC imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ	SBC rel	SBC ind, Y	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX*			SBC abs, X	INC abs, X		F
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes

Note: † = New Address Modes

■ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

		IMME- DIATE	ABSO- LUTE	ZERO PAGE	ACCUM	IM- PLIED	(IND, X)	(IND, Y)	ZPG, X	ZPG, Y	ABS, X	ABS, Y	RELA- TIVE	(ABS)	ABS (IND, X)	(ZPG)	PROCESSOR STATUS CODES										
MNE	OPERATION	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	7	6	5	4	3	2	1	0	MNE		
ADC	A ← A + C + A	(1,3)	69	2	2	6D	4	3	65	3	2						72	5	2	N	V		Z	C	ADC		
AND	A ← A & A	(1)	29	2	2	2D	4	3	25	3	2						32	5	2	N			Z	C	AND		
ASL	[E] ← [E] × 2	(1)				OE	6	3	06	5	2	0A	2	1											ASL		
BCC	Branch if C=0	(2)																							BCC		
BCS	Branch if C=1	(2)																							BCS		
BEQ	Branch if Z=1	(2)																							BEQ		
BIT	A ← A & M	(4,5)	89	2	2	2C	4	3	24	3	2														BIT		
BMI	Branch if N=1	(2)																							BMI		
BNE	Branch if Z=0	(2)																							BNE		
BPL	Branch if N=0	(2)																							BPL		
BRA	Branch Always	(2)																							BRA		
BRK	Break								00	7	1														BRK		
BVC	Branch if V=0	(2)																							BVC		
BVS	Branch if V=1	(2)																							BVS		
CLC	0 ← C								18	2	1														CLC		
CLD	0 ← D																								CLD		
CLI	0 ← I																								CLI		
CLV	0 ← V																								CLV		
CMP	A ← M	(1)	C9	2	2	CD	4	3	C5	3	2														CMP		
CPX	X ← M		E0	2	2	EC	4	3	E4	3	2														CPX		
CPY	Y ← M		C0	2	2	CC	4	3	C4	3	2														CPY		
DEA	A ← 1 + A																								DEA		
DEC	M ← 1 + M	(1)				CE	6	3	C6	5	2														DEC		
DEX	X ← 1 + X																								DEX		
DEY	Y ← 1 + Y																								DEY		
EOR	A ← A ⊕ M		49	2	2	4D	4	3	45	3	2														EOR		
INA	A ← 1 + A																								INA		
INC	M ← 1 + M	(1)				EE	6	3	E6	5	2														INC		
INX	X ← 1 + X																								INX		
INY	Y ← 1 + Y																								INY		
JMP	Jump to new loc					4C	3	3																	JMP		
JSR	Jump Subroutine					20	6	3																	JSR		
LDA	M ← A	(1)	A9	2	2	AD	4	3	A5	3	2														LDA		
LDX	M ← X	(1)	A2	2	2	AE	4	3	A6	3	2														LDX		
LDY	M ← Y	(1)	A0	2	2	AC	4	3	A4	3	2														LDY		
LSR	[E] ← [E] ÷ 2	(1)				4E	6	3	46	5	2	4A	2	1											LSR		
NOP	PC ← 1 + PC																								NOP		
ORA	A ← A ∨ M	(1)	09	2	2	0D	4	3	05	3	2														ORA		
PHA	A ← M ₆ S 1 + S																								PHA		
PHP	P ← M ₆ S 1 + S																								PHP		
PHX	X ← M ₆ S 1 + S																								PHX		
PHY	Y ← M ₆ S 1 + S																								PHY		
PLA	S ← 1 + S M ₆ ← A																								PLA		
PLP	S ← 1 + S M ₆ ← P																								PLP		
PLX	S ← 1 + S M ₆ ← X																								PLX		
PLY	S ← 1 + S M ₆ ← Y																								PLY		
ROL	[E] ← [E] × 2	(1)				2E	6	3	2E	5	2	2A	2	1											ROL		
ROR	[E] ← [E] ÷ 2	(1)				6E	6	3	6E	5	2	6A	2	1											ROR		
RTI	Return from Inter.																								RTI		
RTS	Return from Subr.																								RTS		
SBC	A ← A - C + A	(1,3)	E9	2	2	ED	4	3	E5	3	2														SBC		
SEC	1 ← C																								SEC		
SED	1 ← D																								SED		
SEI	1 ← I																								SEI		
STA	A ← M					8D	4	3	85	3	2														STA		
STX	X ← M					8E	4	3	86	3	2														STX		
STY	Y ← M					8C	4	3	84	3	2														STY		
STZ	00 ← M					9C	4	3	94	3	2														STZ		
TAX	A ← X																								TAX		
TAY	A ← Y																								TAY		
TRB	A ← A & M	(4)				1C	6	3	14	5	2														TRB		
TSB	A ← A & M	(4)				0C	6	3	04	5	2														TSB		
TSX	S ← X																								TSX		
TXA	X ← A																								TXA		
TXS	X ← S																								TXS		
TYA	Y ← A																								TYA		

Notes:

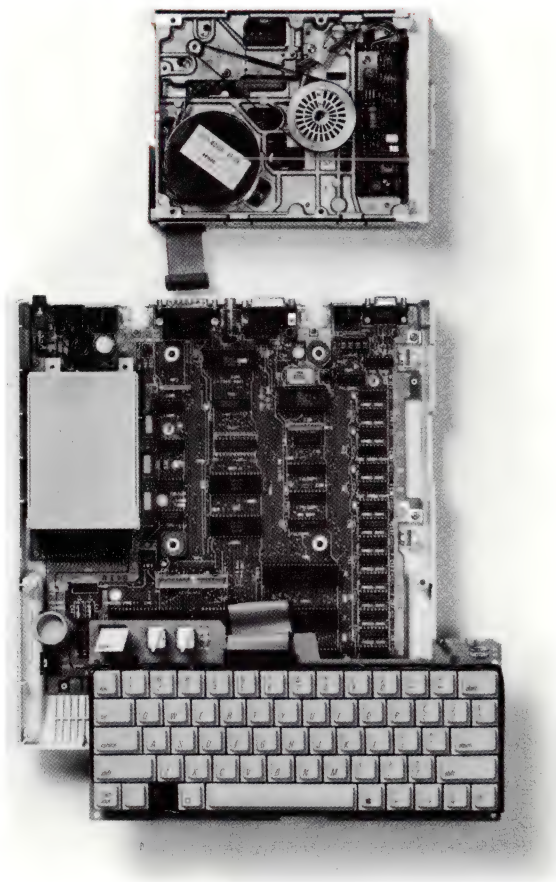
- Add 1 to "n" if page boundary is crossed.
- Add 1 to "n" if branch occurs to same page.
Add 2 to "n" if branch occurs to different page.
- Add 1 to "n" if decimal mode.
- V bit equals memory bit 6 prior to execution.
N bit equals memory bit 7 prior to execution.

X Index X
 Y Index Y
 A Accumulator
 M Memory per effective address
 Ms Memory per stack pointer

+ Add
 - Subtract
 ∧ And
 V Or
 ⊕ Exclusive or

n No. Cycles
 # No. Bytes
 M₆ Memory bit 6
 M₇ Memory bit 7

- *5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.



This appendix lists all important RAM and hardware locations in address order and briefly describes them. It also provides cross-references to the section of the manual where they are described further. Appendix C contains a similar list for important firmware addresses.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32767) the complementary decimal value for use in Apple Integer BASIC.

B.1 Page \$00

Table B-1 lists the zero page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- M denotes the Monitor.
- A denotes Applesoft BASIC.
- I denotes Integer BASIC.
- D denotes DOS 3.3.
- P denotes ProDOS. Locations whose contents ProDOS saves and restores afterward have a P in parentheses, indicating that ProDOS has no net effect on them.

Table B-1. Page \$00 Use

Hex	Dec	Used by	Hex	Dec	Used by
\$00	0	A	\$10	16	A
\$01	1	A	\$11	17	A
\$02	2	A	\$12	18	A
\$03	3	A	\$13	19	A
\$04	4	A	\$14	20	A
\$05	5	A	\$15	21	A
\$06	6		\$16	22	A
\$07	7		\$17	23	A
\$08	8		\$18	24	A
\$09	9		\$19	25	
\$0A	10	A	\$1A	26	
\$0B	11	A	\$1B	27	
\$0C	12	A	\$1C	28	
\$0D	13	A	\$1D	29	
\$0E	14	A	\$1E	30	
\$0F	15	A	\$1F	31	

Hex	Dec	Used by		Hex	Dec	Used by	
\$20	32	M		\$48	72	M	D (P)
\$21	33	M		\$49	73	M	(P)
\$22	34	M		\$4A	74		I D (P)
\$23	35	M		\$4B	75		I D (P)
\$24	36	M		\$4C	76		I D (P)
\$25	37	M		\$4D	77		I D (P)
\$26	38	M	D	\$4E	78	M	(P)
\$27	39	M	D	\$4F	79	M	
\$28	40	M	D				
\$29	41	M	D	\$50	80	M	A
\$2A	42	M	D	\$51	81	M	A
\$2B	43	M	D	\$52	82	M	A
\$2C	44	M	D	\$53	83	M	A
\$2D	45	M	D	\$54	84	M	A
\$2E	46	M	D	\$55	85	M	A I
\$2F	47	M	D	\$56	86		A I
				\$57	87		A I
\$30	48	M		\$58	88		A I
\$31	49	M		\$59	89		A I
\$32	50	M		\$5A	90		A I
\$33	51	M		\$5B	91		A I
\$34	52	M		\$5C	92		A I
\$35	53	M	D	\$5D	93		A I
\$36	54	M	D	\$5E	94		A I
\$37	55	M	D	\$5F	95		A I
\$38	56	M	D				
\$39	57	M	D	\$60	96		A I
\$3A	58	M	P	\$61	97		A I
\$3B	59	M	P	\$62	98		A I
\$3C	60	M	P	\$63	99		A I
\$3D	61	M	P	\$64	100		A I
\$3E	62	M	D P	\$65	101		A I
\$3F	63	M	D P	\$66	102		A I
				\$67	103		A I D
\$40	64	M	D (P)	\$68	104		A I D
\$41	65	M	D (P)	\$69	105		A I D
\$42	66	M	D (P)	\$6A	106		A I D
\$43	67	M	D (P)	\$6B	107		A I
\$44	68	M	D (P)	\$6C	108		A I
\$45	69	M	D (P)	\$6D	109		A I
\$46	70	M	D (P)	\$6E	110		A I
\$47	71	M	D (P)	\$6F	111		A I D

Table B-1—continued. Page \$00 Use

Hex	Dec	Used by	Hex	Dec	Used by
\$70	112	A I D	\$98	152	A I
\$71	113	A I	\$99	153	A I
\$72	114	A I	\$9A	154	A I
\$73	115	A I	\$9B	155	A I
\$74	116	A I	\$9C	156	A I
\$75	117	A I	\$9D	157	A I
\$76	118	A I	\$9E	158	A I
\$77	119	A I	\$9F	159	A I
\$78	120	A I			
\$79	121	A I	\$A0	160	A I
\$7A	122	A I	\$A1	161	A I
\$7B	123	A I	\$A2	162	A I
\$7C	124	A I	\$A3	163	A I
\$7D	125	A I	\$A4	164	A I
\$7E	126	A I	\$A5	165	A I
\$7F	127	A I	\$A6	166	A I
			\$A7	167	A I
\$80	128	A I	\$A8	168	A I
\$81	129	A I	\$A9	169	A I
\$82	130	A I	\$AA	170	A I
\$83	131	A I	\$AB	171	A I
\$84	132	A I	\$AC	172	A I
\$85	133	A I	\$AD	173	A I
\$86	134	A I	\$AE	174	A I
\$87	135	A I	\$AF	175	A I D
\$88	136	A I			
\$89	137	A I	\$B0	176	A I D
\$8A	138	A I	\$B1	177	A I
\$8B	139	A I	\$B2	178	A I
\$8C	140	A I	\$B3	179	A I
\$8D	141	A I	\$B4	180	A I
\$8E	142	A I	\$B5	181	A I
\$8F	143	A I	\$B6	182	A I
			\$B7	183	A I
\$90	144	A I	\$B8	184	A I
\$91	145	A I	\$B9	185	A I
\$92	146	A I	\$BA	186	A I
\$93	147	A I	\$BB	187	A I
\$94	148	A I	\$BC	188	A I
\$95	149	A I	\$BD	189	A I
\$96	150	A I	\$BE	190	A I
\$97	151	A I	\$BF	191	A I

Table B-1—continued. Page \$00 Use

Hex	Dec	Used by	Hex	Dec	Used by
\$C0	192	A I	\$E0	224	A
\$C1	193	A I	\$E1	225	A
\$C2	194	A I	\$E2	226	A
\$C3	195	A I	\$E3	227	
\$C4	196	A I	\$E4	228	A
\$C5	197	A I	\$E5	229	A
\$C6	198	A I	\$E6	230	A
\$C7	199	A I	\$E7	231	A
\$C8	200	A I	\$E8	232	A
\$C9	201	A I	\$E9	233	A
\$CA	202	A I D	\$EA	234	A
\$CB	203	A I D	\$EB	235	
\$CC	204	A I D	\$EC	236	
\$CD	205	A I D	\$ED	237	
\$CE	206	I	\$EE	238	
\$CF	207	I	\$EF	239	
\$D0	208	A I	\$F0	240	A
\$D1	209	A I	\$F1	241	A
\$D2	210	A I	\$F2	242	A
\$D3	211	A I	\$F3	243	A
\$D4	212	A I	\$F4	244	A
\$D5	213	A I	\$F5	245	A
\$D6	214	I	\$F6	246	A
\$D7	215	I	\$F7	247	A
\$D8	216	A I D	\$F8	248	A
\$D9	217	A I	\$F9	249	
\$DA	218	A I	\$FA	250	
\$DB	219	A I	\$FB	251	
\$DC	220	A I	\$FC	252	
\$DD	221	A I	\$FD	253	
\$DE	222	A I	\$FE	254	
\$DF	223	A I	\$FF	255	

B.2 Page \$03

Most of page \$03 is available for small machine-language programs. The built-in Monitor uses the top 16 addresses of page \$03, as shown in Figure B-2; the XFer routine (Section 2.5.3) uses locations \$03ED and \$03EE. If you are using DOS or ProDOS, it also uses the 32 locations \$03D0 through \$03EF.

Table B-2. Page \$03 Use

Hex	Dec	Section	Use
\$03F0	1008	2.6.4	Address of BRK request handler (normally \$59, \$FA)
\$03F1	1009		
\$03F2	1010	2.6.4, 10.1	Reset vector
\$03F3	1011		
\$03F4	1012	2.6.4	Power-up byte (see text)
\$03F5	1013	10.6.4	Jump instruction to Applesoft &-command handler (initially \$4C, \$58, \$FF)
\$03F6	1014		
\$03F7	1015		
\$03F8	1016		
\$03F9	1017	10.6.4	Jump instruction to user CONTROL-Y command handler
\$03FA	1018		
\$03FB	1019		
\$03FC	1020	2.6.4	Address of user IRQ interrupt handler
\$03FD	1021		
\$03FE	1022		
\$03FF	1023		

B.3 Screen Holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of 8 memory addresses are left over in 16 areas of the text and low-resolution display pages—8 areas in main RAM and 8 in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4.

Table B-3. Main Memory Screen Hole Allocations

Hex	Dec	Section	Description
\$0478	1144	9.1.5	Mouse port: low byte of clamping minimum
\$0479	1145	7.5	Reserved for serial port 1
\$047A	1146	8.5	Reserved for serial port 2
\$047B	1147		Reserved
\$047C	1148	9.1.5	Low byte of X coordinate
\$047D	1149		Reserved for mouse port
\$047E	1150		Reserved
\$047F	1151		Reserved
\$04F8	1272	9.1.5	Mouse port: low byte of clamping maximum
\$04F9	1273	7.5, E.6.3	Reserved for serial port 1
\$04FA	1274	8.5, E.6.2	Reserved for serial port 2
\$04FB	1275		Reserved
\$04FC	1276	9.1.5	Low byte of Y coordinate
\$04FD	1277		Reserved for mouse port
\$04FE	1278		Reserved
\$04FF	1279	E.6.4	Reserved
\$0578	1400	9.1.5	Mouse port: high byte of clamping minimum
\$0579	1401	7.5	Port 1 printer width (1-255; 0 = unlimited)
\$057A	1402	8.5	Port 2 line length (1-255; 0 = unlimited)
\$057B	1403		Cursor horizontal position (80-column display)
\$057C	1404	9.1.5	High byte of X coordinate
\$057D	1405		Reserved for mouse port
\$057E	1406		Reserved
\$057F	1407	E.6.4	Reserved
\$05F8	1528	9.1.5	Mouse port: high byte of clamping maximum
\$05F9	1529	7.5, E.6.3	Port 1 temporary storage location
\$05FA	1530	8.5, E.6.2	Port 2 temporary storage location
\$05FB	1531		Reserved
\$05FC	1532	9.1.5	High byte of Y coordinate
\$05FD	1533		Reserved for mouse port
\$05FE	1534		Reserved
\$05FF	1535	E.6.2	Reserved

Table B-3—continued. Main Memory Screen Hole Allocations

Hex	Dec	Section	Description
\$0678	1656		Reserved
\$0679	1657	7.5	Indicates when port 1 firmware is parsing a command
\$067A	1658	8.5	Indicates when port 2 firmware is parsing a command
\$067B	1659		Reserved
\$067C	1660	9.1.5	Mouse port: reserved
\$067D	1661		Reserved for mouse port
\$067E	1662		Reserved
\$067F	1663	E.6.4	Reserved
\$06F8	1784		Reserved
\$06F9	1785	7.5	Current port 1 command character
\$06FA	1786	8.5	Current port 2 command character
\$06FB	1787		Reserved
\$06FC	1788	9.1.5	Mouse port: reserved
\$06FD	1789		Reserved for mouse port
\$06FE	1790		Reserved
\$06FF	1791	E.6.2	Reserved
\$0778	1912		DEVNO: \$n0 = current active port number x 16
\$0779	1913	7.5	Port 1 flags for echo and auto line feed
\$077A	1914	8.5	Port 2 flags for each and auto line feed
\$077B	1915		Reserved
\$077C	1916	9.1.5, E.6.1	Mouse port status byte
\$077D	1917		Reserved for mouse port
\$077E	1918		Reserved
\$077F	1919		Reserved
\$07F8	2040		MSLOT: owner of \$C800–\$CFFF (\$C3, video)
\$07F9	2041	7.5	Port 1 current printer column
\$07FA	2042	8.5	Port 2 current line position
\$07FB	2043		Reserved
\$07FC	2044	9.1.5	Mouse port mode byte
\$07FD	2045		Reserved for mouse port
\$07FE	2046		Reserved
\$07FF	2047		Reserved

Table B-4. Auxiliary Memory Screen Hole Allocations

Hex	Dec	Section	Description
\$0478	1144	7.5	Initial port 1 ACIA control register values (\$9E)
\$0479	1145	7.5	Initial port 1 ACIA command register values (\$0B)
\$047A	1146	7.5	Initial port 1 characteristics flags (\$40)
\$047B	1147	7.5	Initial port 1 printer width (\$50)
\$047C	1148	8.5	Initial port 2 ACIA control register values (\$16)
\$047D	1149	8.5	Initial port 2 ACIA command register values (\$0B)
\$047E	1150	8.5	Initial port 2 characteristics flags (\$01)
\$047F	1151	8.5	Initial port 2 line length (\$00)
\$04F8 through \$04FF	1272 1279		Reserved
\$0578 through \$057F	1400 1407		Reserved
\$05F8 through \$05FF	1528 1535		Reserved
\$0678 through \$067F	1656 1663		Reserved
\$06F8 through \$06FF	1784 1791		Reserved
\$0778 through \$077F	1912 1919		Reserved
\$07F8 through \$07FF	2040 2047		Reserved

B.4 The Hardware Page

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc. These tables have a column at the left that is not present in other tables. This column, labeled *RW*, indicates the action to take at a particular location.

- R means read.
- RR means read twice in succession.
- R7 means read the byte and then check bit 7; in the use column, see if... refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of \$80, so if the contents of the location are greater than or equal to \$80, the bit is on.

Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).

- R/W means to either read or write. For writing, the value is unimportant.
- W means to write only. The value is unimportant.
- N means not to read or write, because the location is reserved.

An address of the form \$C00x refers to the 16 locations from \$C000 through \$C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, others do not. Your program will have to assign a label for it anyway.

Table B-5. Addresses \$C000-\$C03F

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	\$C00x			KStrb	4.1	Read keyboard data (bits 0–6) and strobe (bit 7)
W	\$C000	49152	–16384	80Store	2.5.4, 5.6	Off: Page2 switches Page 1 and 2
W	\$C001	49153	–16383	80Store	2.5.4, 5.6	On: Page2 switches Page 1 and 1X
W	\$C002	49154	–16382	RAMRd	2.5.2	Off: Read main 48K RAM
W	\$C004	49156	–16380	RAMWrt	2.5.2	Off: Write in main 48K RAM
W	\$C005	49157	–16379	RAMWrt	2.5.2	On: Write in auxiliary 48K RAM
W	\$C006	49158	–16378			Reserved
W	\$C007	49159	–16377			Reserved
W	\$C008	49160	–16376	AltZP	2.4.2	Off: Use main P0, P1, bank-switched RAM
W	\$C009	49161	–16375	AltZP	2.4.2	On: Use auxiliary P0, P1, bank-switched RAM
W	\$C00A	49162	–16374			Reserved
W	\$C00B	49163	–16373			Reserved
W	\$C00C	49164	–16372	80Col	5.6	Off: 40-column display
W	\$C00D	49165	–16371	80Col	5.6	On: 80-column display
W	\$C00E	49166	–16270	AltChar	5.6	Off: Display primary character set
W	\$C00F	49167	–16369	AltChar	5.6	On: Display alternate character set
W	\$C01x				4.1	Clear keyboard strobe (\$C00x bit 7)
R7	\$C010	49168	–16368	AKD	4.1	See if any key now down; clear strobe
R7	\$C011	49169	–16367	RdBnk2	2.4.2	See if using \$D000 bank 2 (or 1)
R7	\$C012	49170	–16366	RdLCRAM	2.4.2	See if reading RAM (or ROM)
R7	\$C013	49171	–16365	RdRAMRd	2.5.2	See if reading auxiliary 48K RAM (or main)
R7	\$C014	49172	–16364	RdRAMWrt	2.5.2	See if writing auxiliary 48K RAM (or main)
R	\$C015	49173	–16363	RstXInt	9.1.3	Reset mouse X0 interrupt
R7	\$C016	49174	–16362	RdAltZP	2.4.2	See if auxiliary P0, P1 and bank-switched RAM
R	\$C017	49175	–16361	RstYInt	9.1.3	Reset mouse Y interrupt
R7	\$C018	49176	–16360	Rd80Store	2.5.4, 5.6	See if 80Store on (or off)
R7	\$C019	49177	–16359	RstVBI	9.1.3	See if VBIInt off (1); reset it
R7	\$C01A	49178	–16358	RdTEXT	5.6	See if text (or graphics)
R7	\$C01B	49179	–16357	RdMIXED	5.6	See if mixed mode switch on
R7	\$C01C	49180	–16356	RdPage2	2.5.4, 5.6	See if Page 2/1X selected (or 1)
R7	\$C01D	49181	–16355	RdHiRes	2.5.4, 5.6	See if high-resolution switch on
R7	\$C01E	49182	–16354	RdAltChar	5.6	See if alternate character set (or primary)
R7	\$C01F	49183	–16353	Rd80Col	5.6	See if 80-column hardware on
N	\$C020	49184	–16352			Reserved (read and write)
N	through \$C02F	49199	–16337			
W	\$C030	49200	–16336	Reserved	4.2.1	Toggle speaker
R	\$C030	49200	–16336			
N	\$C031	49201	–16335			Reserved (read and write)
N	through \$C03F	49215	–16321			

Table B-6. Addresses \$C040–\$C05F

RW	Hex	Dec	Neg Dec	Label	Section	Use
R7	\$C040	49216	—16320	RdXYMsk	9.1.3	See if X0/Y0 mask set
R7	\$C041	49217	—16319	RdVBIMsk	9.1.3	See if VBI mask set
R7	\$C042	49218	—16318	RdX0Edge	9.1.3	See if interrupt on falling X0 edge
R7	\$C043	49219	—16317	RdY0Edge	9.1.3	See if interrupt on falling Y0 edge
N	\$C044	49220	—16316			Reserved
N	\$C045	49221	—16315			Reserved
N	\$C046	49222	—16314			Reserved
N	\$C047	49223	—16313			Reserved
R	\$C048	49224	—16312	RstXY	9.1.3	Reset X0/Y0 interrupt flags
N	\$C049	49225	—16311			Reserved
N	\$C04A	49226	—16310			Reserved
N	\$C04B	49227	—16309			Reserved
N	\$C04C	49228	—16308			Reserved
N	\$C04D	49229	—16307			Reserved
N	\$C04E	49230	—16306			Reserved
N	\$C04F	49231	—16305			Reserved
R/W	\$C050	49232	—16304	TEXT	5.6	Off: Graphics display
R/W	\$C051	49233	—16303	TEXT	5.6	On: Text display
R/W	\$C052	49234	—16302	MIXED	5.6	Off: Text or graphics only
R/W	\$C053	49235	—16301	MIXED	5.6	On: Combination text and graphics
R/W	\$C054	49236	—16300	Page2	2.5.4, 5.6	Off: Use Page 1
R/W	\$C055	49237	—16299	Page2	2.5.4, 5.6	On: Display Page 2 (80Store off); store to Page 1X (80Store on)
R/W	\$C056	49238	—16298	HiRes	2.5.4, 5.6	Off: Low-resolution
R/W	\$C057	49239	—16297	HiRes	2.5.4, 5.6	On: High-resolution; double if 80Col and DHiRes on
N	\$C058	49240	—16296			Reserved if IOUDis on (\$C07E bit 7=1)
R/W	\$C059	49241	—16295	DisXY	9.1.3	Disable (mask) mouse X0/Y0 interrupts
N	\$C05A	49242	—16294			Reserved if IOUDis on
R/W	\$C05B	49243	—16293	EnbXY	9.1.3	Enable (allow) mouse X0/Y0 interrupts
N	\$C05C	49244	—16292			Reserved if IOUDis on
R/W	\$C05D	49245	—16291	DisVBI	9.1.3	Disable (mask) VBI interrupts
N	\$C05E	49246	—16290			Reserved if IOUDis on
R/W	\$C05F	49247	—16289	EnVBI	9.1.3	Enable (allow) VBI interrupts
N	\$C060	49248	—16288			Reserved if IOUDis on
R/W	\$C061	49249	—16287	X0Edge	9.1.3	Interrupt on rising edge of X0
N	\$C062	49250	—16286			Reserved if IOUDis on
R/W	\$C063	49251	—16285	X0Edge	9.1.3	Interrupt on falling edge of X0
R/W	\$C064	49252	—16284	DHiRes	5.6	If IOUDis on: Set double-high-resolution
R/W	\$C065	49253	—16283	Y0Edge	9.1.3	If IOUDis off: Interrupt on rising Y0
R/W	\$C066	49254	—16282	DHiRes	5.6	If IOUDis on: Clear double-high-resolution
R/W	\$C067	49255	—16281	Y0Edge	9.1.3	If IOUDis off: Interrupt on falling Y0

Table B-7. Addresses \$C060–\$C07F

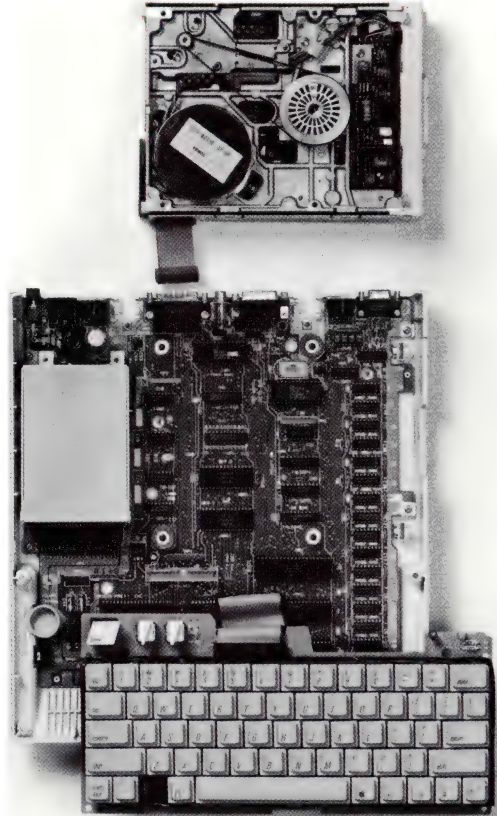
RW	Hex	Dec	Neg Dec	Label	Section	Use
W	\$C06x					Reserved (write)
R7	\$C060	49248	— 16288	Rd80Sw	4.1	See if 80/40 switch down (= 40)
R7	\$C061	49249	— 16287	RdBtn0	4.1, 9.1.3	See if mouse button/☐ pressed
R7	\$C062	49250	— 16286	RdBtn1	4.1, 9.2	See if switch 1/☐ pressed
R7	\$C063	49251	— 16285	Rd63	9.1, 9.2	See if mouse button not pressed
R7	\$C064	49252	— 16284	Pdl0	9.2	See if hand control button 0 pressed
R7	\$C065	49253	— 16283	Pdl1	9.2	See if hand control button 1 pressed
R7	\$C066	49254	— 16282	MouX1	9.1.3	See if mouse X1 (direction) is high
R7	\$C067	49255	— 16281	MouY1	9.1.3	See if mouse Y1 (direction) is high
N	\$C068	49256	— 16280			
	through					Reserved (write and read)
N	\$C06F	49263	— 16273			
R/W	\$C07x					Trigger paddle timer; reset VBIInt; however, some \$C07x are reserved
R/W	\$C070	49264	— 16272	PTrig	9.2	Designated trigger or reset location
N	\$C071	49265	— 16271			
	through					Reserved
N	\$C07D	49277	— 16259			
R7	\$C07E	49278	— 16258	RdIOUDis		See if IOUDis on; trigger paddle timer; reset VBIInt
W				IOUDis	5.6, 9.1.3	On: Enable access to DHiRes switch; disable \$C058–\$C05F IOU access
R7	\$C07F	49279	— 16257	RdDHiRes	5.6, 9.1.3	See if DHiRes on
W				IOUDis	5.6	Off: Disable access to DHiRes switch; enable \$C058–\$C05F IOU access

Table B-8. Addresses \$C080–\$C0AF

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	\$C080	49280	–16256		2.4.2	Read RAM; no write; use \$D000 bank 2
RR	\$C081	49281	–16255		2.4.2	Read ROM, write RAM; use \$D000 bank 2
R	\$C082	49282	–16254		2.4.2	Read ROM; no write; use \$D000 bank 2
RR	\$C083	49283	–16253		2.4.2	Read and write RAM; use \$D000 bank 2
N	\$C084	49284	–16252			Reserved
N	\$C085	49285	–16251			Reserved
N	\$C086	49286	–16250			Reserved
N	\$C087	49287	–16249			Reserved
R	\$C088	49288	–16248		2.4.2	Read RAM; no write; use \$D000 bank 1
RR	\$C089	49289	–16247		2.4.2	Read ROM, write RAM; use \$D000 bank 1
R	\$C08A	49290	–16246		2.4.2	Read ROM; no write; use \$D000 bank 1
RR	\$C08B	49291	–16245		2.4.2	Read and write RAM; use \$D000 bank 1
N	\$C08C	49292	–16244			Reserved
N	\$C08D	49293	–16243			Reserved
N	\$C08E	49294	–16242			Reserved
N	\$C08F	49295	–16241			Reserved
N	\$C090 through	49296	–16240			Reserved
N	\$C097	49303	–16233			
R/W	\$C098	49304	–16232		7.3, 11.11	Port 1 ACIA transmit/receive register
R/W	\$C099	49305	–16231		7.3, 11.11	Port 1 ACIA status register
R/W	\$C09A	49306	–16230		7.3, 11.11, E	Port 1 ACIA command register
R/W	\$C09B	49307	–16229		7.3, 11.11	Port 1 ACIA control register
N	\$C09C through	49308	–16228			Reserved
N	\$C09F	49311	–16225			
N	\$C0A0 through	49312	–16224			Reserved
N	\$C0A7	49319	–16217			
R/W	\$C0A8	49320	–16216		8.3, 11.11	Port 2 ACIA transmit/receive register
R/W	\$C0A9	49321	–16215		8.3, 11.11	Port 2 ACIA status register
R/W	\$C0AA	49322	–16214		8.3, 11.11, E	Port 2 ACIA command register
R/W	\$C0AB	49323	–16213		8.3, 11.11	Port 2 ACIA control register
N	\$C0AC through	49324	–16212			Reserved
N	\$C0AF	49327	–16209			

Table B-9. Addresses \$C0B0–\$C0FF

RW	Hex	Dec	Neg Dec	Label	Section	Use
N	\$C0B0 through	49328	— 16208			Reserved
N	\$C0BF	49343	— 16193			
N	\$C0C0 through	49344	— 16192			Reserved
N	\$C0CF	49359	— 16177			
N	\$C0D0 through	49360	— 16176			Reserved
N	\$C0DF	49375	— 16161			
N	\$C0E0 through	49376	— 16160			Reserved
N	\$C0EF	49391	— 16145			
N	\$C0F0 through	49392	— 16144			Reserved
N	\$C0FF	49407	— 16129			



This appendix lists all significant firmware addresses: entry points, locations containing the addresses of entry points, and locations where machine and device identification bytes reside.

▲ **Warning**

The Monitor firmware entry points are the only **published** entry points in the sense that they are the only ones that will remain in the same locations in future Apple II series computers.

The firmware protocol identification bytes and offsets will work with other Apple II series computers only if used as directed (Section 3.4.2).

C.1 The Tables

This appendix supplements the chapter text by specifying three forms of each address: hexadecimal, decimal, and complementary (negative) decimal.

In these tables, some of the addresses are followed by a label. These labels are listed only to help you find the named location in the firmware listings, or to remember the function found at the address. The Apple IIc contains no global label table: your program must assign its own labels to the addresses as required.

There are several types of information at these firmware addresses: actual entry points (labeled *entry*), the low-order byte of an entry point (labeled *offset*), a device or machine identification byte (labeled *ident*), indicators (labeled *indic*) specifying whether there are optional routines, vector addresses (labeled *vector*), or an RTS instruction location.

The column labeled *Section* contains the number of the section that describes the item. If there is no description except in a table in this appendix, a section number is not listed.

Each input/output port has an associated protocol table, as shown in Tables C-1 through C-4. Many of the bytes (labeled *offset*) in the protocol tables are the low-order bytes of addresses of I/O routines for the ports; the high-order byte of these addresses must be \$Cn (where n is the port number). This structure is explained in Chapter 3. Although your program must perform some extra processing to use these tables, the benefit is simplified compatible port and slot I/O for all Apple II series machines.

C.2 Port Addresses

Addresses for serial ports 1 and 2, output port 3, and mouse input port 4 are shown in the following four tables.

Table C-1. Serial Port 1 Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C100	49408	— 16128		entry	3.1.1	Main port 1 entry point
\$C105	49413	— 16123		ident	3.4.2	ID byte (\$38)
\$C107	49415	— 16121		ident	3.4.2	ID byte (\$18)
\$C10B	49419	— 16117		ident	3.4.2	Firmware card signature (\$01)
\$C10C	49420	— 16116		ident	3.4.2	Super Serial Card ID (\$31)
\$C10D	49421	— 16115		offset	7.4	Low-order PInit address
\$C10E	49422	— 16114		offset	7.4	Low-order PRead address
\$C10F	49423	— 16113		offset	7.4	Low-order PWrite address
\$C110	49424	— 16112		offset	7.4	Low-order PStatus address
\$C111	49425	— 16111		indic	3.4.2	Non-zero: no optional routines

Table C-2. Serial Port 2 Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C200	49664	— 15872		entry	3.1.1	Main port 2 entry point
\$C205	49669	— 15867		ident	3.4.2	ID byte (\$38)
\$C207	49671	— 15865		ident	3.4.2	ID byte (\$18)
\$C20B	49675	— 15861		ident	3.4.2	Firmware card ID (\$01)
\$C20C	49676	— 15860		ident	3.4.2	Super Serial Card ID (\$31)
\$C20D	49677	— 15859		offset	8.4	Low-order PInit address
\$C20E	49678	— 15858		offset	8.4	Low-order PRead address
\$C20F	49679	— 15857		offset	8.4	Low-order PWrite address
\$C210	49680	— 15856		offset	8.4	Low-order PStatus address
\$C211	49681	— 15855		indic	3.4.2	Non-zero: no optional routines

Table C-3. Video Firmware Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C300	49920	— 15616		entry	3.1.1	Main video entry point (out put only)
\$C305	49925	— 15611	C3KeyIn	ident	3.4.2	ID byte (\$38)
\$C307	49927	— 15609	C3COut1	ident	3.4.2	ID byte (\$18)
\$C30B	49931	— 15605		ident	3.4.2	Firmware card signature (\$01)
\$C30C	49932	— 15604		ident	3.4.2	80-column card ID (\$88)
\$C30D	49933	— 15603		offset	5.9	Low-order PInit address
\$C30E	49934	— 15602		offset	5.9	Low-order PRead address
\$C30F	49935	— 15601		offset	5.9	Low-order PWrite address
\$C310	49936	— 15600		offset	5.9	Low-order PStatus address
\$C311	49937	— 15599	MoveAux	entry	2.5.3	Routine for main/auxiliary control swapping (also called AuxMove)

Table C-4. Mouse Port Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C400	50176	— 15360		entry		Main mouse entry point
\$C405	50181	— 15355		ident	3.4.2	ID byte (\$38)
\$C407	50183	— 15353		ident	3.4.2	ID byte (\$18)
\$C40B	50187	— 15349		ident	3.4.2	Firmware card signature (\$01)
\$C40C	50188	— 15348		type	3.4.2	X-Y pointing device ID (\$20)
\$C40D	50189	— 15347		offset	9.1.4	Low-order PInit address
\$C40E	50190	— 15346		offset	9.1.4	Low-order PRead address
\$C40F	50191	— 15345		offset	9.1.4	Low-order PWrite address
\$C410	50192	— 15344		offset	9.1.4	Low-order PStatus address
\$C411	50193	— 15343		indic	3.4.2	Optional routines follow (\$00)
\$C412	50194	— 15342	SetMouse	offset	9.1.4	Low-order SetMouse address
\$C413	50195	— 15341	ServeMouse	offset	9.1.4	Low-order ServeMouse address
\$C414	50196	— 15340	ReadMouse	offset	9.1.4	Low-order ReadMouse address
\$C415	50197	— 15339	ClearMouse	offset	9.1.4	Low-order ClearMouse address
\$C416	50198	— 15338	PosMouse	offset	9.1.4	Low-order PosMouse address
\$C417	50199	— 15337	ClampMouse	offset	9.1.4	Low-order ClampMouse address
\$C418	50200	— 15336	HomeMouse	offset	9.1.4	Low-order HomeMouse address
\$C419	50201	— 15335	InitMouse	offset	9.1.4	Low-order InitMouse address

C.3 Other Video and I/O Firmware Addresses

Miscellaneous firmware addresses are listed in Table C-5.

Table C-5. Apple IIc Enhanced Video and Miscellaneous Firmware

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C600	50688	—14848		entry	6.1	Disk drive firmware entry point
\$C700	50944	—14592		entry	6.2	External disk startup routine
\$C803	51203	—14333	NewIRQ	entry	E.1	IRQ handling routine

C.4 Applesoft BASIC Interpreter Addresses

The addresses of Applesoft BASIC entry points are listed in the *Applesoft BASIC Programmer's Reference Manual*. The Applesoft interpreter occupies ROM addresses from \$D000 through \$F7FF.

C.5 Monitor Addresses

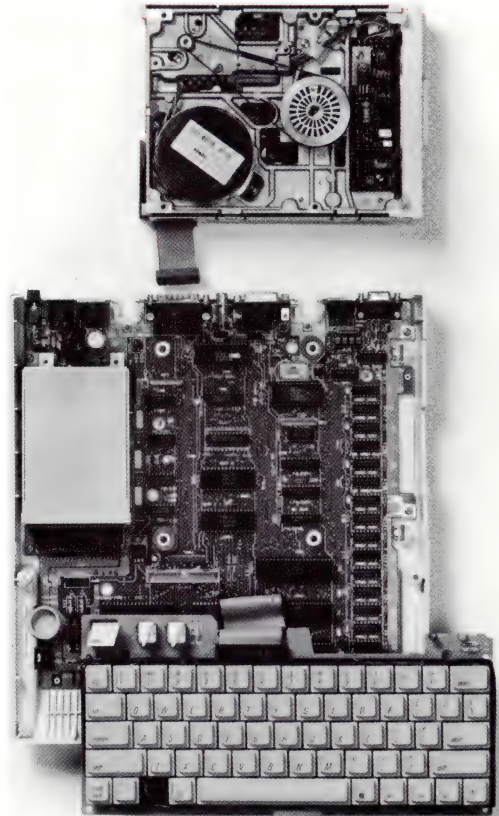
Table C-6 lists the Monitor entry points, machine identifier bytes, interrupt vectors, and the address of a known RTS instruction.

Table C-6. Apple IIc Monitor Entry Points and Vectors

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$F800	63488	—2048	PLOT	entry	5.8	Plots a low-resolution block
\$F819	63513	—2023	HLine	entry	5.8	Draws low-resolution horizontal line
\$F828	63528	—2008	VLine	entry	5.8	Draws low-resolution vertical line
\$F832	63538	—1998	ClrScr	entry	5.8	Clears low-resolution screen
\$F836	63542	—1994	ClrTop	entry	5.8	Clears top 40 low-resolution lines
\$F864	63588	—1948	SetCol	entry	5.8	Sets low-resolution color (Table 5-4)
\$F871	63601	—1935	SCRN	entry	5.8	Reads color of low-resolution block
\$F941	63809	—1727	PrntAX	entry	5.8	Displays A and X in hex
\$F94A	63818	—1718	PrBI2	entry	5.8	Sends X blanks to output
\$FA47	63845	—1691	NewBRK	entry	E.2	Apple IIc break handler
\$FA62	64098	—1438	Reset	entry	2.6	Hardware reset routine

Table C-6—continued. Apple IIc Monitor Entry Points and Vectors

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$FB1E	64386	—1150	PRead	entry	9.2	Reads hand controller position
\$FB6F	64467	—1169	SetPwrC	entry	2.6.4	Routine to create power-up byte
\$FBB3	64535	—1101		ident	F.1.2	Machine identification byte
\$FBC0	64548	—1088		ident	F.1.2	Machine identification byte
\$FBDD	64477	—1059	Bell1	entry	4.2.2	Sends 1-kHz beep to speaker
\$FC42	64578	—958	ClrEOP	entry	5.8	Clears from cursor to bottom
\$FC58	64600	—936	HOME	entry	5.8	Clears from cursor to upper left
\$FC9C	64668	—868	ClrEOL	entry	5.8	Clears from cursor to end of line
\$FC9E	64670	—866	ClEOLZ	entry	5.8	Clears from BASL to end of line
\$FCA8	64680	—856	WAIT	entry		Delays for time specified by A
\$FD0C	64780	—756	RdKey	entry	3.2.1	Displays cursor, jumps to KSW
\$FD1B	64795	—741	KeyIn	entry	3.2.2	Waits for keypress, reads key
\$FD35	64821	—715	RdChar	entry	4.1.2	Gets input, interprets ESC codes
\$FD67	64871	—665	GetLnZ	entry	4.1.2	Sends CR to output, goes to GetLn
\$FD6A	64874	—662	GetLn	entry	3.2.3	Displays prompt, gets input line
\$FD6F	64879	—657	GetLn1	entry	4.1.2	No prompt; gets input line
\$FD8B	64907	—629	CROut1	entry	5.8	Clears to end of line, calls CROut
\$FD8E	64910	—626	CROut	entry	5.8	Sends CR to output
\$FDDA	64986	—550	PrByte	entry	5.8	Sends A to output
\$FDE3	64995	—541	PrHex	entry	5.8	Displays low nibble of A in hex
\$FDED	65005	—531	COut	entry	3.3.1	Jumps to CSW
\$FDF0	65008	—528	COut1	entry	3.3.2	Displays A, advances cursor
\$FE2C	65068	—468	MOVE	entry		Copies memory elsewhere
\$FE36	65078	—458	VERIFY	entry		Compares two blocks of memory
\$FF2D	65325	—211	PrErr	entry	5.8	Sends ERR to output; beeps
\$FF3A	65338	—198	Bell	entry	4.2.2	Sends CONTROL-G to output
\$FF3F	65343	—193	IORest	entry		Loads \$45–\$49 into registers
\$FF4A	65354	—182	IOSave	entry		Stores A, X, Y, P, S at \$45–\$49
\$FF58	65368	—168	IORTS	RTS		Location of known RTS instruction
\$FF69	65385	—151	Monitor	entry	10.1	Standard Monitor entry point
\$FFFA	65530	—6		vector		Low-order NMI vector (unused)
\$FFFB	65531	—5		vector		High-order NMI vector (unused)
\$FFFC	65532	—4		vector		Low-order reset vector (\$62)
\$FFFD	65533	—3		vector		High-order reset vector (\$FA)
\$FFFE	65534	—2	IRQVect	vector		Low-order IRQ vector (\$03)
\$FFFF	65535	—1		vector		High-order IRQ vector (\$CB)



This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIc. It is not intended to be a complete description. For more information, refer to the manuals that are provided with each product.

D.1 Operating Systems

This section discusses the operating systems that the Apple IIc works with. CP/M, and any other operating system that requires an interface card, does not work on the Apple IIc.

D.1.1 ProDOS

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive (on original Apple IIc's with the command PR#7), interrupts, and all other hardware and firmware features of the Apple IIc.

D.1.2 DOS

The Apple IIc works with DOS 3.3. Its built-in disk drive hardware and firmware can also access DOS 3.2 disks by using the *BASIC*S disk. DOS support is provided for the sake of Apple II series compatibility; neither version of DOS takes full advantage of all the features of the Apple IIc.

D.1.3 Pascal Operating System

Versions 1.2 and later of the Pascal Operating System use the 80/40 switch and the interrupt features of the Apple IIc, while remaining compatible with the other Apple II series computers.

While the Apple IIc works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc does not work with Pascal 1.0, because the I/O firmware entry points of that version of the operating system are rigidly defined (rather than being accessed via a table), and the Apple IIc's built-in firmware does not correspond to these entry points.

D.2 Languages

This section discusses using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

D.2.1 Applesoft BASIC

The programming examples in this manual are almost entirely in assembly language, and so most addresses and values are given in hexadecimal notation.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. The values used by Applesoft must be in decimal, so you will have to convert hexadecimal values given in this manual to decimal. (Several tables in this manual include decimal equivalents to make the job easier for you.)

If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

D.2.2 Integer BASIC

You will have to run a version of DOS in your Apple IIc to use Integer BASIC. ProDOS does not support Integer BASIC.

D.2.3 Pascal

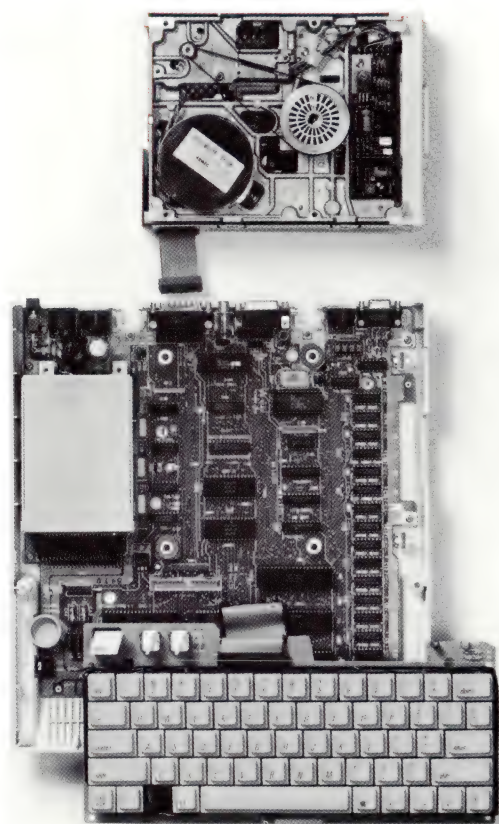
The Pascal language runs on the Apple IIc under versions 1.1 or later of the Pascal Operating System. However, for best performance, use Pascal versions 1.2 or later.

D.2.4 FORTRAN

FORTRAN runs under version 1.1 of the Pascal Operating System, which, as explained in Section D.1.3, does not detect or use certain Apple IIc features, such as the 80/40 switch or auxiliary memory. Therefore, FORTRAN does not take advantage of these features either.

D.2.5 Logo II

Apple Logo II works under ProDOS on Apple II series machines with at least 128K of memory. Logo II is a version of the Logo language originally developed from the LISP (LISt Processing) language at MIT as a language to be used for learning. Logo II takes advantage of the Apple II's graphics and retains much of the power and flavor of LISP without LISP's somewhat cryptic syntax.



This appendix describes the sources of interrupts on the Apple IIc, how the firmware handles the interrupts, and how to use interrupt-driven features directly in those rare cases when the firmware cannot meet your needs.

▲Warning

If you use interrupt hardware directly, instead of using the built-in interrupt-handling firmware, you can't be sure that your programs will be compatible with possible future Apple II series computers or revisions.

E.1 Introduction

This section describes interrupts and their effects on the Apple IIc hardware.

E.1.1 What Is an Interrupt?

An interrupt is a signal that a computer uses to know when to stop what it's doing so it can quickly handle a time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This lets the system keep track of the mouse's position and maintain smooth movement of the pointer on the screen.

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the computer preserve a *snapshot* of its state when interrupted, so that when it continues later with what it was doing, those conditions can be restored.

E.1.2 Interrupts on Apple II Computers

Interrupts have not always been fully supported on the Apple II. All versions of Apple's DOS, as well as the Monitor program, rely on the integrity of location \$45, which the built-in interrupt handler has always destroyed by saving the accumulator in it. Most versions of Pascal simply do not work with interrupts enabled.

The Apple IIc built-in interrupt handler now saves the accumulator on the stack instead of in location \$45. DOS 3.3, ProDOS, Pascal 1.2 (or later versions) and the Monitor all work with interrupts on the Apple IIc.

You should use either ProDOS or Pascal 1.2 (or later versions) if you want interrupt-using software to work on the Apple IIe and the Apple II Plus. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time since interrupts that occur while interrupts are disabled cannot be detected. Due to the critical timing of disk read and write operations, Pascal, DOS 3.3, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all the interrupt sources discussed below are turned off.

On the Apple IIe only, interrupts are periodically turned off while 80-column screen operations are being performed. This is most noticeable while the screen is scrolling. Also, most peripheral cards used in the Apple IIe disable interrupts while reading and writing.

E.1.3 Interrupt Handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. The IRQ line on the microprocessor can be pulled low if 65C02 interrupts are not masked (that is, the CLI instruction has been used). This is the standard technique that devices use when they need immediate attention.
2. The processor executes a break (BRK, opcode \$00) instruction.
3. A nonmaskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens on the Apple IIc.

The first two possibilities cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in \$FFFE and \$FFFF. The sequence performed by the 65C02 is:

1. If an IRQ occurs, finish executing the current instruction. (If a BRK occurs, the current instruction is already finished.)
2. Push the high byte of the program counter onto stack.
3. Push the low byte of the program counter onto stack.
4. Push the program status byte onto stack.
5. Jump to the address stored in \$FFFE, \$FFFF—that is, JMP (\$FFFE).

The different sources of interrupt signals are discussed below.

E.1.4 The Interrupt Vector at \$FFFE

In the Apple IIc there are three separate regions of memory that contain address \$FFFE: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at \$FFFE in the ROM points to Apple IIc's built-in interrupt handling routine. You should generally use the built-in interrupt handler, rather than writing your own, because of the complexity of interrupts on the Apple IIc.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

E.2 The Built-in Interrupt Handler

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, it decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The built-in interrupt handling routine records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

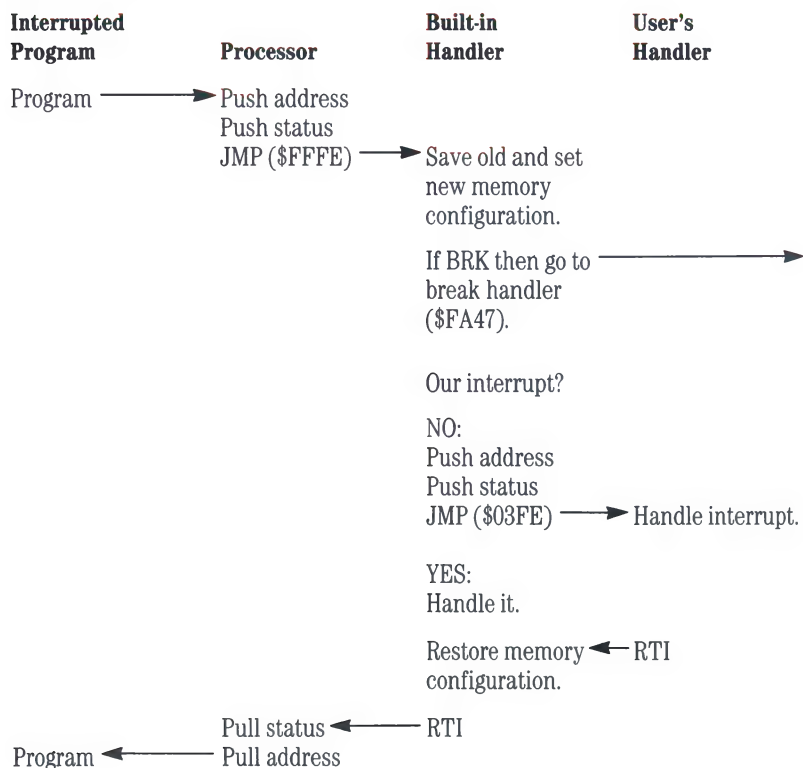
Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described in Section E.4.

If the interrupt was not caused by a BRK, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts, or passes them to a user's interrupt handling routine whose address is stored at \$03FE and \$03FF of main memory.

After handling an interrupt itself, or after the user's handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does an RTI to restore processing to where it was when the interrupt occurred. Figure E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

Figure E-1. Interrupt-Handling Sequence



E.2.1 Saving the Memory Configuration

The built-in interrupt handler saves the state of the system, and sets it to a known state according to these rules:

- If 80Store and Page2 are on, then it switches in text Page 1 (Page2 off) so that main screen holes are accessible.
- It switches in main memory for reading (RAMRd off).
- It switches in main memory for writing (RAMWrt off).
- It switches in ROM addresses \$D000-\$FFFF for reading (RdLCRAM off).
- It switches in main stack and zero page (AltZP off).
- It preserves the auxiliary stack pointer, and restores the main stack pointer (see Section E.2.2).
- It preserves the current ROM state and switches in the ROM bank 1.

Note: Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

E.2.2 Managing Main and Auxiliary Stacks

Because the Apple IIc has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: \$0100 for the main stack pointer when the auxiliary stack is active, and \$0101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address \$0100, and initialize the auxiliary stack pointer to \$FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at \$0101, and the main stack pointer is retrieved from \$0100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

E.3 User's Interrupt Handler at \$03FE

You can set up screen hole locations to indicate that the user's interrupt handler should be called when certain interrupts occur. To do this, place your interrupt handler's address at \$03FE and \$03FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return with an RTI.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored four bytes down on the stack. This byte is explained in Section E.4.

In general there is no guaranteed *maximum* response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80Store and Page2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page \$02 switched in for reading and writing).

E.4 Handling Break Instructions

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode \$00) instruction. (If it was, bit 4 of the processor status byte is a 1.) If so, it jumps to a break handling routine, which saves the state of the computer at the time of the break as follows:

Information	Location
Program counter (low byte)	\$3A
Program counter (high byte)	\$3B
Encoded memory state	\$44
Accumulator	\$45
X register	\$46
Y register	\$47
Status register	\$48

Finally, the break routine jumps to the routine whose address is stored at \$03F0 and \$03F1.

The encoded memory state in location \$44 can be interpreted as follows:

Bit 7	=	0	
Bit 6	=	1	if 80Store and Page2 both on

Bit 5	=	1	if auxiliary RAM switched in for reading
Bit 4	=	1	if auxiliary RAM switched in for writing
Bit 3	=	1	if bank-switched RAM being read
Bit 2	=	1	if bank-switched \$D000 page \$01 switched in
Bit 1	=	1	if bank-switched \$D000 page \$02 switched in
Bit 0	=	0	

E.5 Sources of Interrupts

The Apple IIc can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic sources of interrupts: use of the mouse, and actions affecting the two 6551 ACIA circuits (the chips that control serial communication). How to use these sources of interrupts in conjunction with the built-in interrupt handler is discussed in Section E.6.

Mouse use can cause interrupts when

- ☐ the mouse is moved in the horizontal (X) direction
- ☐ the mouse is moved in the vertical (Y) direction
- ☐ the mouse button is pressed.

Interrupts can also be generated every 1/60 second by the rising edge of the vertical blanking signal. This is called the vertical blanking (VBL) interrupt, and is synchronized with a signal used for the video display.

Actions affecting the ACIA circuits can cause interrupts when

- ☐ a key is pressed (the firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user)
- ☐ either ACIA has received a byte of data from its port (the firmware can use this interrupt to buffer data or it can pass the interrupt on to the user)
- ☐ pin 5 of either serial port changes state (device ready/not ready to accept data) (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)
- ☐ either ACIA is ready to accept another character to be transmitted (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)

- the keyboard strobe is cleared (the firmware absorbs this interrupt).

An interrupt can also be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.

E.6 Firmware Handling of Interrupts

The following sections discuss how the various sources of interrupts should be used together with the built-in interrupt handler.

E.6.1 Firmware for Mouse and VBL

As described in Chapter 9, the mouse can be initialized (by the SetMouse call) to nine different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

When the mouse is initialized, the interrupt vector is copied to addresses \$FFFE and \$FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are those listed in Section E.5 as resulting from mouse use.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SetMouse call). If so, the user's interrupt handler, whose address is stored at \$03FE, is called.

The user's interrupt handler should first call ServeMouse to determine the source of the interrupt. This call updates the mouse status byte at \$077C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at \$077C are as follows:

Bit	1 means that
3	Interrupt was from vertical blanking
2	Interrupt was from button
1	Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to ReadMouse. This causes the mouse coordinates and status to be updated as follows:

\$047C	Low byte of X coordinate
\$04FC	Low byte of Y coordinate
\$057C	High byte of X coordinate
\$05FC	High byte of Y coordinate
\$077C	Button and movement status

Bit	Means
7	0 = button up; 1 = button down
6	0 = button up on last ReadMouse 1 = button down on last ReadMouse
5	0 = no movement since last ReadMouse 1 = movement since last ReadMouse
3-1	Always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI.

Remember that interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler will absorb the interrupts. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses \$FFFE and \$FFFF in bank-switched RAM. Interrupts will be ignored whenever the \$D000-\$FFFF ROM is switched in.

E.6.2 Firmware for Keyboard Interrupts

The Apple IIc hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page \$08 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RdKey (\$FD0C).

▲Warning

Do not call the buffer reading routine directly. Its entry address will not be the same in future versions of the computer.

The special characters `CONTROL-S` (stop list) and `CONTROL-C` (stop Applesoft execution) do not work while keyboard buffering is turned on. A new keystroke, `APPLE CONTROL-X`, clears the buffer.

Using Keyboard Buffering Firmware

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself this way:

1. Disable processor interrupts (SEI).
2. Set location \$05FA to \$80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations \$05FF and \$06FF to \$80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

If you are using the serial ports at the same time, just set the low bit of \$C0AA to 1. This prevents receiver interrupts from being turned off.

A PR#2 or IN#2 or the equivalent will shut off keyboard interrupts.

5. Enable processor interrupts (CLI).

Using Keyboard Interrupts Through Firmware

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$05FA to \$C0. This tells the firmware to identify a keystroke interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location \$04FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$04FA to \$00.

E.6.3 Using External Interrupts Through Firmware

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1. It can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$05F9 to \$C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of \$C09A to the value \$0F. For example:

LDA \$C09A	Read port 1 ACIA command register
ORA #\$0F	Set low nibble to \$0F
STA \$C09A	Set port 1 ACIA command register

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location \$04F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$04F9 to \$00.

E.6.4 Firmware for Serial Interrupts

The Apple IIc hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, data are ignored. Only one port can be buffered at a time. The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 8.

Using Serial Buffering Transparently

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

1. Disable processor interrupts (SEI).
2. Set location \$04FF to \$C1 to buffer port 1, or to \$C2 to buffer port 2.
3. Set locations \$057F and \$067F to \$00. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$0D. For example:

LDA \$C09A	Read port 1 ACIA command register
AND \$F0	Clear low nibble
ORA #\$0D	Set low nibble to \$0D
STA \$C09A	Set port 1 ACIA command register

The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

5. Enable processor interrupts (CLI).

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

Using Serial Interrupts Through Firmware

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

1. Disable processor interrupts (SEI).
2. Set location \$04FF to a value other than \$C1 or \$C2.
3. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$0D. For example:

LDA \$C09A	Read port 1 ACIA command register
AND \$F0	Clear low nibble
ORA #\$0D	Set low nibble to \$0D
STA \$C09A	Set port 1 ACIA command register

The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

4. Enable processor interrupts (CLI).

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware (\$04F9 for port 1; \$04FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte (\$04F9 for port 1; \$04FA for port 2).

Transmitting Serial Data

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.
- The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remain in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

A Loophole in the Firmware

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.

E.7 Bypassing the Interrupt Firmware

The following sections give further details on using interrupts on the Apple IIc computer without using the built-in interrupt handler.

A method of handling mouse interrupts directly is described in Chapter 9.

E.7.1 Using Mouse Interrupts Without the Firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If the \$D000-\$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts.

Tables E-1 and E-2 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on when done (CLI).

Table E-1. Activating Mouse Interrupts

To Activate Interrupts On	Enable IOU Access	Select Source	Enable Source	Disable IOU Access
Mouse X (rising edge)	STA \$C079	STA \$C05C	STA \$C059	STA \$C078
Mouse X (falling edge)	STA \$C079	STA \$C05D	STA \$C059	STA \$C078
Mouse Y (rising edge)	STA \$C079	STA \$C05E	STA \$C059	STA \$C078
Mouse Y (falling edge)	STA \$C079	STA \$C05F	STA \$C059	STA \$C078
VBL	STA \$C079		STA \$C05B	STA \$C078

Table E-2. Reading Mouse Interrupts

To Read Interrupts From	Read Direction (A.S.A.P.)	Determine Source	Handle It	Return
Mouse X	LDA \$C066	LDA \$C015 (bit 7=1 if true)		RTI
Mouse Y	LDA \$C067	LDA \$C017 (bit 7=1 if true)		RTI
VBL		LDA \$C019 (bit 7=1 if true)		RTI

The mouse direction data read from \$C066 and \$C067 are guaranteed valid for at least 40 microseconds, and average duration is at least 200 microseconds, so you should read the direction as soon as possible.

E.7.2 Using ACIA Interrupts Without the Firmware

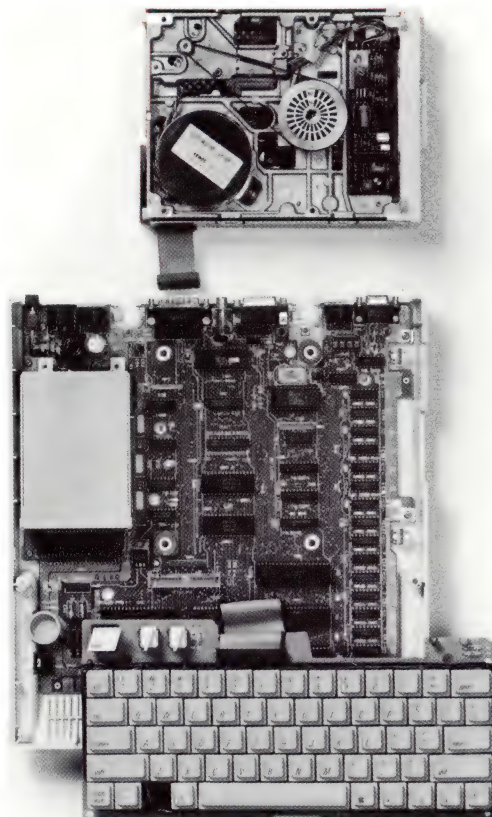
To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the \$D000–\$FFFF ROM is ever switched in, the built-in interrupt handler will handle the interrupt as determined by certain mode bytes (Section E.6.1).

When writing your serial interrupt handler, refer to Figures 11-31 through 11-33 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIAs have the following connections:

- Port 1:** DSR line connected to the EXTINT line on the external disk port.
 DCD line connected to pin 5 of port 1 connector.
- Port 2:** DSR line goes high when a key is pressed.
 DCD line connected to pin 5 of port 2 connector.

The ACIA registers have the following addresses:

Port 1:		Port 2:	
Data register	= \$C098	Data register	= \$COA8
Status register	= \$C099	Status register	= \$COA9
Command register	= \$C09A	Command register	= \$COAA
Control register	= \$C09B	Control register	= \$COAB



This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II. It does not contain an exhaustive list of differences, but it does mention those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more Apple II series models.

F.1 Overview

The differences between the Apple II series computers can be expressed as a series of equations: this computer equals that one plus or minus certain features.

Note: The following equations compare each model of Apple II series with its predecessor in terms of functional equivalence, not literal equality. For example,

Apple II Plus = Apple II — Integer BASIC firmware

does not mean that Integer BASIC firmware can be removed from the Apple II—just that the one machine functions as if it were the other without such firmware.

Apple II Plus	=	II	<ul style="list-style-type: none"> + Autostart ROM + Applesoft firmware + 48K RAM standard — old Monitor ROM — Integer BASIC firmware
Apple IIe	=	II Plus	<ul style="list-style-type: none"> + Apple Language Card (with 16K of RAM) + 80-column (enhanced) video firmware + built-in diagnostics + full ASCII keyboard + internal power light + FCC approval + improved back panel + 9-pin back panel game connector + auxiliary slot (with possibility of 80-column text card and extra 64K RAM)

Apple IIc

= IIe

- slot 0
- + interrupt support in firmware (enhanced IIe)
- + Mini-Assembler in firmware (enhanced IIe)
- + extended 80-column text card
- + 80/40 switch
- + keyboard switch
- + disk-use light
- + disk controller port
- + disk drive
- + mouse port
- + serial printer port
- + serial communication port
- + built-in port firmware
- + video expansion connector
- removable cover
- slots 1 to 7
- auxiliary slot
- internal power light
- cassette I/O connectors
- internal game I/O connector (hence no game output)
- auxiliary video pin
- Monitor cassette support
- + Mini-Assembler in firmware (IIc with UniDisk 3.5 support)
- + Protocol Converter in firmware (IIc with UniDisk 3.5 support)

F.1.1 Type of Processor

The processor in the Apple II and II Plus is the 6502. The original Apple IIe uses a 6502A. The Apple IIc and enhanced Apple IIe both use the 65C02: this is a redesigned CMOS CPU that has 27 new instructions, new addressing modes, and for some instructions a differing execution scheme and machine cycle counts (see Appendix A).

Programs written for the Apple IIc will run on the earlier machines only if they do not contain instructions unique to the 65C02, or depend on shared instructions whose cycle times differ. Programs should also use only published entry points in the Monitor firmware to allow maximum compatibility between different Apple II series computers.

F.1.2 Machine Identification

Identification of Apple II series computers is as shown in Table F-1.

Table F-1. Apple II Series Identification Bytes

Machine	\$FBB3	\$FB1E	\$FBC0	\$FBBF
Apple II	\$38			
Apple II Plus	\$EA	\$AD		
Apple IIe	\$06		\$EA	
Apple IIe (enhanced)	\$06		\$E0	
Apple IIc	\$06		\$00	\$FF
Apple IIc (UniDisk 3.5 support)	\$06		\$00	\$00
Apple III in Apple II emulation mode	\$EA	\$8A		

Any future Apple II series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Vendor Technical Support.

The MachID byte for ProDOS (\$BF98 on the global page) will have bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and to 1 if the computer is not one of these machines. In an Apple IIc, bits 7 and 6 are also set to binary 10.

Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

F.2 Memory Structure

This section compares the memory organization of the Apple IIc with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

F.2.1 Amount and Address Ranges of RAM

The Apple II could have as little as 4K of RAM at the time of purchase, and could be upgraded to as much as 48K of RAM, following a procedure described in the *Apple II Reference Manual*.

The Apple II Plus has 48K of RAM (\$0000 through \$BFFF) as a standard feature. With the addition of an Apple Language Card, a 48K Apple II or II Plus could be expanded to have 64K of RAM.

The Apple IIe has a full 64K of RAM. The top 12K addresses overlap with the ROM addresses \$D000 through \$FFFF. There is an additional bank-switched area of 4K from \$D000 through \$DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the \$Dxxx banks by changing soft switches located in memory.


With an Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64K of RAM available, although no more than half of the 128K of RAM space is available at any given time. Soft switches located in memory control these address space selections.

The RAM in the Apple IIc is equivalent to the RAM in an Apple IIe with an Extended 80-Column Card.

F.2.2 Amount and Address Ranges of ROM

The Apple II has 8K of ROM (\$E000 through \$FFFF), and the Apple II Plus has 12K of ROM (\$D000 through \$FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from \$D000 through \$FFFF.

The Apple IIe has 16K of ROM, of which it uses 15.75 K (addresses \$C100 through \$FFFF; page \$C0 addresses are for I/O hardware). ROM addresses \$C300 through \$C3FF (normally assigned to the ROM in a card in slot 3) and \$C800 through \$CFFF contain 80-column video firmware; ROM addresses \$C100 through \$C2FF and \$C400 through \$C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6, and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with -CONTROL-RESET causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75 K from \$C100 through \$FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not preempt any port firmware space.

UniDisk 3.5

The Apple IIc with built-in UniDisk 3.5 support has twice the ROM (32K) of the original Apple IIc. The extra ROM contains support for the Protocol Converter, a Mini-Assembler, STEP and TRACE functions in the Monitor firmware, expanded self-test routines, and improved interrupt support.

In the Apple IIc, addresses \$C100 through \$CFFF contain I/O and interrupt firmware, addresses \$D000 through \$F7FF contain the Applesoft BASIC interpreter, and addresses \$F800 through \$FFFF contain the Monitor.

F.2.3 Peripheral-Card Memory Spaces

Each Apple IIc port has up to 16 peripheral-card I/O space locations in main memory on the hardware page (beginning at location \$C0s0 + \$80 for slot or port s), allocated in the standard Apple II series way (that is, beginning at location \$C0s0 + \$80 for each slot s).

The peripheral-card ROM space (page \$Cs for slot s in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port s, but port routines are not limited to their allocated \$Cs pages.

The 2K-byte expansion ROM space from \$C800 to \$CFFF in the Apple IIc is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space (or scratch-pad RAM) (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc firmware that these locations not be altered by software except for the specific purposes described in Chapters 7, 8, and 9, and in Appendix E.

F.2.4 Hardware Addresses

The hardware page (the addresses from \$C000 through \$C0FF) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc on the one hand, and the Apple II, II Plus,

and IIe on the other, with respect to hardware page use. However, for many characteristics, the Apple IIe and IIC work one way, while the Apple II and II Plus work another.

\$C000–\$C00F

On all Apple II series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIC, writing to each of these addresses turns memory and display switches on and off. Writing to addresses \$C006, \$C007, \$C00A, and \$C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved on the Apple IIC.

For reading the keyboard, use \$C000; reserve \$C001 through \$C00F.

\$C010–\$C01F

On all Apple II series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIC, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use \$C010; reserve \$C011 through \$C01F.

Reading \$C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XInt) on the Apple IIC. Similarly, reading \$C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YInt) on the Apple IIC.

Reading \$C019 checks the current state of vertical blanking (VBL) on the Apple IIe, but it resets the latched vertical blanking interrupt (VBIInt) on the Apple IIC.

\$C020–\$C02F

On the Apple II, II Plus, and IIe, reading any address \$C02x toggles the cassette output signal. On the original Apple IIC, both reading from and writing to these locations are reserved. The Apple IIC with 32K of ROM uses \$C028 to switch in or out the extra 16K of ROM.

\$C030–\$C03F

On all Apple II series computers, reading an address of the form \$C03x toggles the speaker. For full Apple II series compatibility, toggle the speaker using \$C030, and reserve \$C031 through \$C03F.

On the Apple IIC, writing to \$C031 through \$C03F is explicitly reserved.

\$C040–\$C04F

On the Apple II, II Plus, and IIe, reading any address of the form \$C04x triggers the utility strobe. The Apple IIc has no utility strobe.

On the Apple IIc, addresses \$C044 through \$C047 are explicitly reserved, and reading or writing any address from \$C048 through \$C04F resets both the X and Y mouse interrupts (XInt and YInt).

\$C050–\$C05F

Addresses \$C050 through \$C057 work the same on the Apple IIc as on the Apple IIe: they turn the TEXT, MIXED, Page2, and HiRes switches on and off.

On the Apple IIe, addresses \$C058 through \$C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board or later, an Apple Extended 80-Column Text Card, and a jumper installed on the card, reading locations \$C05E and \$C05F set and clear double-high-resolution display mode.

On the Apple IIc, if the IOUDis switch is on, both reading from and writing to addresses \$C058 through \$C05D are reserved, and addresses \$C05E and \$C05F set and clear the double-high-resolution display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDis switch is off, then addresses \$C058 through \$C05F control various characteristics of mouse and vertical blanking interrupts (Table 9-2).

\$C060–\$C06F

On the Apple IIc, writing to any address of the form \$C06x is reserved, and reading addresses \$C068 through \$C06F is reserved.

Reading addresses \$C061 and \$C062 is the same as on the Apple IIe (switch inputs and Apple keys). Reading addresses \$C064 and \$C065 is the same as on all other Apple II series computers (analog inputs 0 and 1).

On the Apple IIc, address \$C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the shift-key mod (used on Apple II, II Plus, and IIe with some text cards) will find it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location \$C063, it will appear that the shift-key mod is not present.

On the Apple IIc, address \$C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input.

The Apple IIc has two, rather than four, analog (paddle) inputs. Addresses \$C066 and \$C067 are used for reading the mouse X and Y direction bits.

\$C070–\$C07F

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form \$C07x triggers the (analog input) paddle timers.

On the Apple IIc, only address \$C070 is to be used for that one function. Addresses \$C071 through \$C07D are explicitly reserved. The results of reading from or writing to addresses \$C07E and \$C07F are described in Table 5-8.

\$C080–\$C08F

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses \$C084 through \$C087 duplicate the functions of the four addresses preceding them, and addresses \$C08C through \$C08F do also. These eight addresses are explicitly reserved on the Apple IIc.

\$C090–\$C0FF

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form \$C080 + \$s0 is allocated to an interface card (if present) in slot s.

On the Apple IIc, addresses corresponding to slots 1, 2, 3, 4, and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.

F.2.5 Monitors

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different; for example, there were no arrow keys for vertical cursor motion.

When you start the Apple IIc with a DOS or *BASIC*S disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along with it. When you type **INT** from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type **FP** to switch back to Applesoft, which uses the new Monitor in ROM, or activate the 80-column firmware.

F.3 I/O in General

Apple IIc I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

F.3.1 DMA Transfers

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc because neither bus is available at any of the back panel connectors.

F.3.2 Slots Versus Ports

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory, and test cards. The Apple IIc has no slots; instead, it has back panel connectors and built-in hardware and firmware that are functional equivalents of slots with cards in them. The back panel connectors are called ports on the Apple IIc.

F.3.3 Interrupts

The Apple IIc is the first computer in the Apple II series to have built-in interrupt-handling capabilities. The enhanced Apple IIe has very similar interrupt-handling capability included.

F.4 Keyboard

Both keyboard layout and character sets vary in the Apple II series computers. The major keyboard difference in the Apple II series is that the Apple IIe and IIc have full ASCII keyboards, while the Apple II and II Plus do not.

F.4.1 Keys, Switches, and Lights

The Apple II and II Plus have identical 52-key keyboards. The Apple IIe and Apple IIc keyboards have the same 63-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and II Plus. While the Apple II and II Plus have a **REPT** key, the IIe and IIc have an auto-repeat feature built into each character key.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not **RESET** works without **CONTROL**. On the Apple IIe and Apple IIc, there is no choice: **CONTROL-RESET** works, and **RESET** alone does not.

The Apple IIc and IIe have an  and a  key; the Apple II and II Plus do not have these two keys.

The captions on several keys—**ESCAPE**, **TAB**, **CONTROL**, **SHIFT**, **CAPS LOCK**, **DELETE**, **RETURN**, and **RESET**—can vary: on the Apple II and II Plus some are abbreviated or missing; on the Apple IIc all keycaps are lowercase italic; on international models, some captions are replaced by symbols (Appendix G).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display; the other is for selecting keyboard layout (Sholes versus Dvorak on USA models) or both keyboard layout and character set (on international models).

The position of the power-on light differs on the Apple II and II Plus, Apple IIe, and Apple IIc. The Apple IIc has a disk-use light as well.

F.4.2 Character Sets

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual*.

The Apple IIe and Apple IIc keyboard character sets are the same: full ASCII. The standard (Sholes) layout and key assignments are described in the *Apple IIe Technical Reference Manual*. The Dvorak layout and key assignments are described in Chapter 4 and Appendix G of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc.

Apple Computer, Inc. manufactures fully localized models (with regard to power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late production models of the same machine. For further details, refer to Appendix G of this manual.

F.5 Speaker

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, audio output jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

F.6 Video Display

This section discusses the general differences between Apple IIc video display capabilities and those of the other computers in the series. Note, however, that as new ROMs become available for the Apple IIe, many differences between these two machines will vanish.

F.6.1 Character Sets

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc and IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the primary character set.

The Apple IIc and IIe have another character set, called the alternate character set, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 5. You can switch character sets at any time by means of the AltChar soft switch, also described in Chapter 5.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be sure of compatibility with some software, you have to switch the Apple IIc keyboard to uppercase by pressing CAPS LOCK.

F.6.2 MouseText

MouseText characters (Chapter 5) are available on every Apple IIc, and on the enhanced Apple IIe.

F.6.3 Vertical Blanking

A signal called vertical blanking indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or II Plus.) On the Apple IIc, vertical blanking is an interrupt (VBLInt) that occurs on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II series computers must take this difference into account.

F.6.4 Display Modes

All models have 40-column text mode, low-resolution graphics mode, high-resolution graphics mode, and mixed graphics and text modes. The Apple IIe (revision B motherboard) with an Apple Extended 80-Column Text Card, and the Apple IIc have double-high-resolution graphics mode also.

F.7 Disk I/O

The Apple II, II Plus, and IIe can support up to six disk drives (although four is the recommended maximum) attached in controller cards plugged into slots 6, 5, and 4. The Apple IIc supports up to two disk drives: its built-in drive (treated as slot 6, drive 1), and one external disk drive (treated as slot 6, drive 2; also treated as slot 7, drive 1 under ProDOS) for external-drive startup purposes.

UniDisk 3.5


The Apple IIc with UniDisk 3.5 support does not use slot 7, drive 1 for external drives. They are handled through the Protocol Converter described in Chapter 6. The firmware for slot 7 (\$C7xx) is needed for other parts of the firmware.

F.8 Serial I/O

The Apple IIc serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

F.8.1 Serial Ports Versus Serial Cards

There are several important differences between Apple IIc serial ports and other Apple II series computers with serial cards installed in them.

Apple IIc serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing -**CONTROL**-**RESET** does not change them.

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc ports also differs from the syntax for serial cards. A separate command character, CONTROL-A or CONTROL-I, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as S instead of B for sending a BREAK signal). Each serial port command letter is unique, to simplify command interpretation.

Changing the command character from CONTROL-A to CONTROL-I, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple IIc serial ports. With the Apple IIc, use the *System Utilities* disk to change modes.

Super Serial Card commands support some functions that Apple IIc serial port commands don't support: translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed, and so on.

UniDisk 3.5

Several new serial port commands are available on the Apple IIc with UniDisk 3.5 support. These commands have been added to make it easier to write programs for the IIc that are also compatible with the Super Serial Card. See Chapters 7 and 8 for these new commands.

Following a CONTROL-I nnnN command, the Apple IIc automatically generates a carriage return after nnnN characters; with the Super Serial Card, you need to turn this on with CONTROL-I C.

F.8.2 Serial I/O Buffers

The communication port firmware uses auxiliary memory page \$08 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also hide data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page \$08. However, such information is destroyed when the communication port is activated.

F.9 Mouse and Hand Controllers

The DB-9 back panel connector on the Apple IIc is used for both the mouse and hand controllers. On the Apple IIe, the DB-9 connector supports hand controllers only; the mouse must use the connector on the interface card.

F.9.1 Mouse Input

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand controller connector. Apple IIc mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc and Apple IIe. However, using the calls in the manner described in Chapter 9 insures mouse support compatibility between the two machines.

The ratio of mouse movement to cursor movement is different on the Apple IIc from on the Apple IIe.

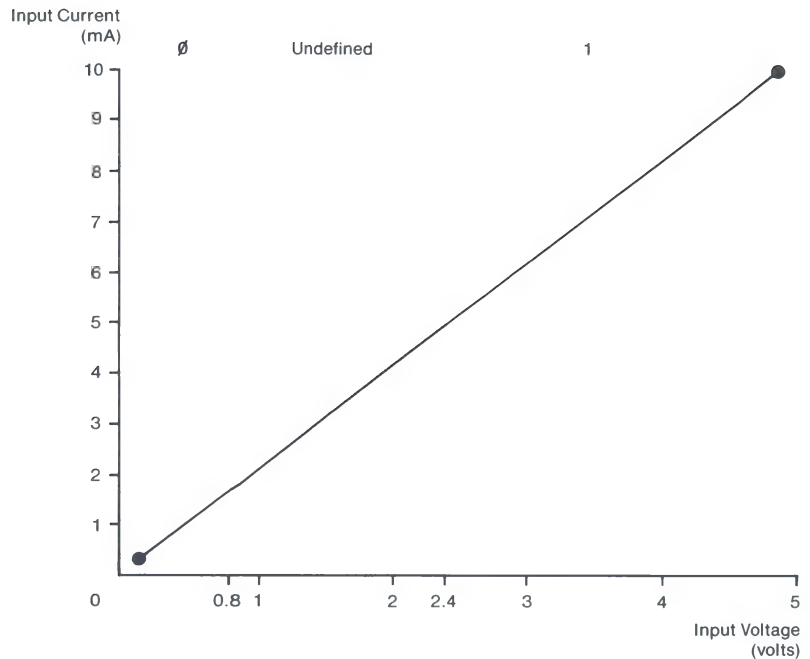
F.9.2 Hand Controller Input and Output

The Apple II, II Plus, and IIe have a 16-pin game I/O connector inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc have a DB-9 back panel connector that supports the three switch inputs and two paddle inputs (plus two more on the internal GAME I/O connector of the Apple II, II Plus, and IIe).

The Apple IIc does not support the four annunciator outputs.

The characteristic response curve for hand controllers differs for the Apple IIc from that of the Apple II, II Plus, and IIe. Compare Figure F-1 with Figure 11-42. This was done so the hardware would support identifiable mouse and hand controller signals using the same circuits.

The paddle timing circuit on the Apple II Plus is slightly different from the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100-ohm fixed resistor is between the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. On the Apple II Plus, the capacitor is connected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.



F.10 Cassette I/O

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and Monitor support. The Apple IIc does not.

UniDisk 3.5

If you plan to run a program on your IIc that handles cassette I/O, make sure that it does not access \$C028. The Apple IIc with UniDisk 3.5 support uses address \$C028 to toggle between its two 16K banks of memory.

F.11 Hardware

Besides the different microprocessors used in various models in the Apple II series (Section F.1.1), there are important differences in power specifications and custom chips.

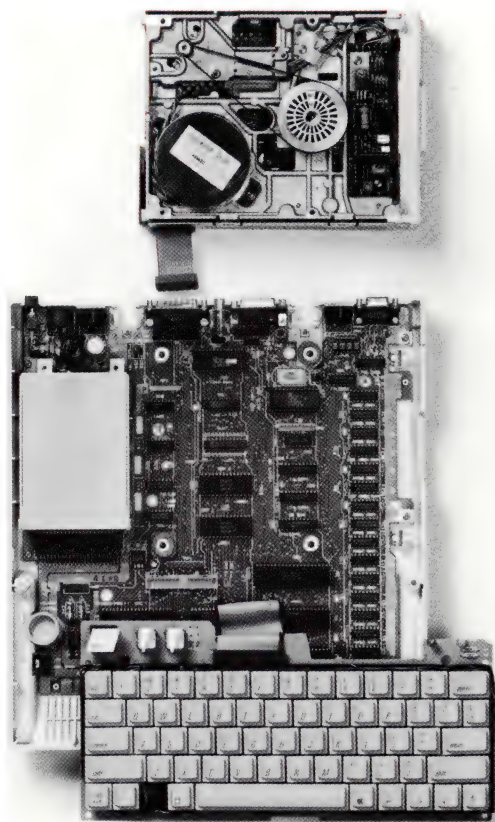
F.11.1 Power

The power supplies for the Apple II, II Plus, and IIe are essentially the same. The floor transformer and voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals.

F.11.2 Custom Chips

The Apple IIe custom chips (memory management unit and input/output unit) replaced dozens of Apple II Plus chips, and added the functionality of dozens more. The Apple IIc has custom MMU and IOU chips, too, but they represent different bonding options, and so their pin assignments are not compatible.

In addition, the Apple IIc has a custom general logic unit (GLU), timing generator (TMG), and disk controller unit (also known as an Integrated Woz Machine, or IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.



This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts, for easy comparison with the other layouts available. Following these is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

G.1 Keyboard Layouts and Codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table G-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table G-7) list only the keystrokes and codes that differ from those in Table G-1.

For example, pressing the **A** key produces *a* (hexadecimal 61); pressing **SHIFT**-**A** produces uppercase *A* (hexadecimal 41); pressing **CONTROL**-**A** or **CONTROL**-**SHIFT**-**A** produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards, from the fact that nothing appears in Tables G-2 through G-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table G-8. In fact, only a few of them change. The pairing of characters on keys varies more.

Note: On all but the French and Italian keyboards, **CAPS LOCK** affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, **CAPS LOCK** down is equivalent to holding down **SHIFT**, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then **CAPS LOCK** does not affect it.

On the French and Italian keyboards, **CAPS LOCK** shifts all the keys. Furthermore, on the French keyboard, when **CAPS LOCK** is down, the **SHIFT** key undoes the shifting.

The shapes and arrangement of keys in Figures G-1 and G-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure G-3 follow the ISO (International Standards Organization) standard used in Europe and elsewhere.

The only differences between the ANSI and ISO versions of the USA keyboard are

- the shapes of three keys: the left `SHIFT` key, `CAPS LOCK`, and `RETURN`
- the resulting repositioning of two keys (`⏏` and `⏏`) in Figures G-1 and G-3
- for some countries, the arrow symbols on `TAB`, `CAPS LOCK`, `RETURN`, and the two `SHIFT` keys (as shown in Figure G-3).

G.1.1 USA Standard (Sholes) Keyboard

Figure G-1 shows the standard (Sholes) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.

Figure G-1. USA Standard or Sholes Keyboard (Keyboard Switch Up)

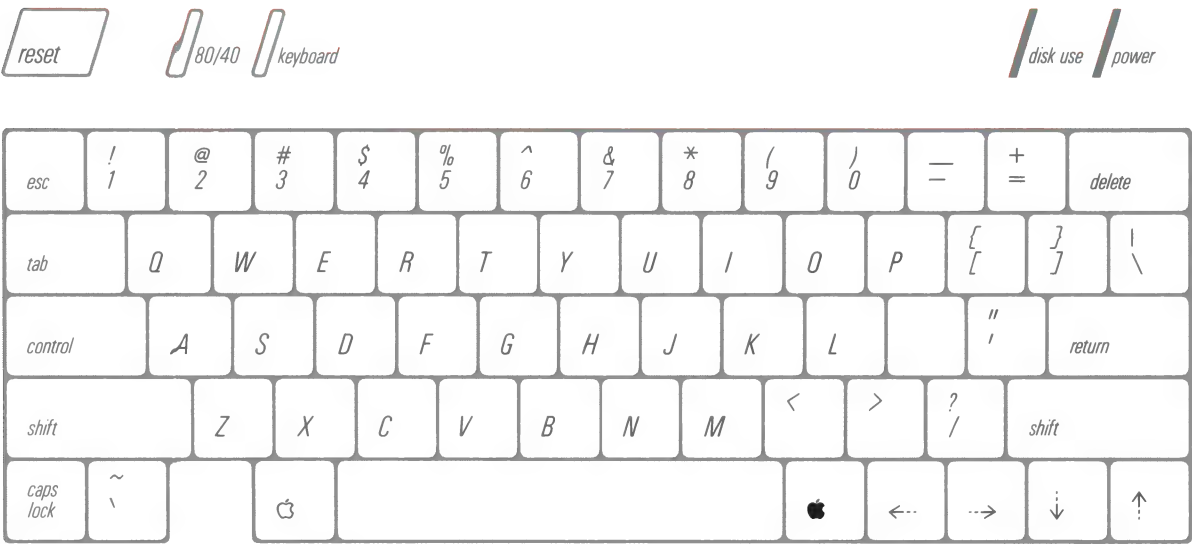


Table G-1. Keys and ASCII Codes

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.





Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
	0A	LF	0A	LF	0A	LF	0A	LF
	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C		2C	,	3C	<	3C	<
- _	2D		1F	US	5F	_	1F	US
. >	2E		2E		3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29)	29)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (39	9	39	9	28	(28	(
	3B		3B		3A		3A	
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] }	5D]	1D	GS	7D	}	1D	GS
` ~	60	`	60		7E	~	7E	

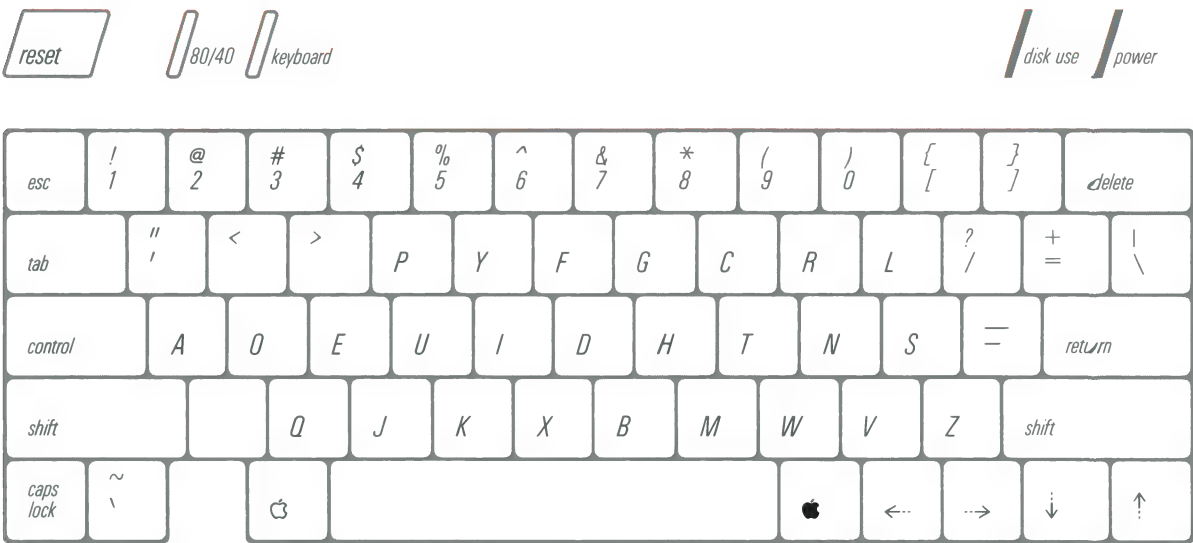
Table G-1—continued. Keys and ASCII Codes*Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.*

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

G.1.2 USA Simplified (Dvorak) Keyboard

Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, and have a working Dvorak keyboard. All keystrokes produce the same ASCII codes as those shown in Table G-1.

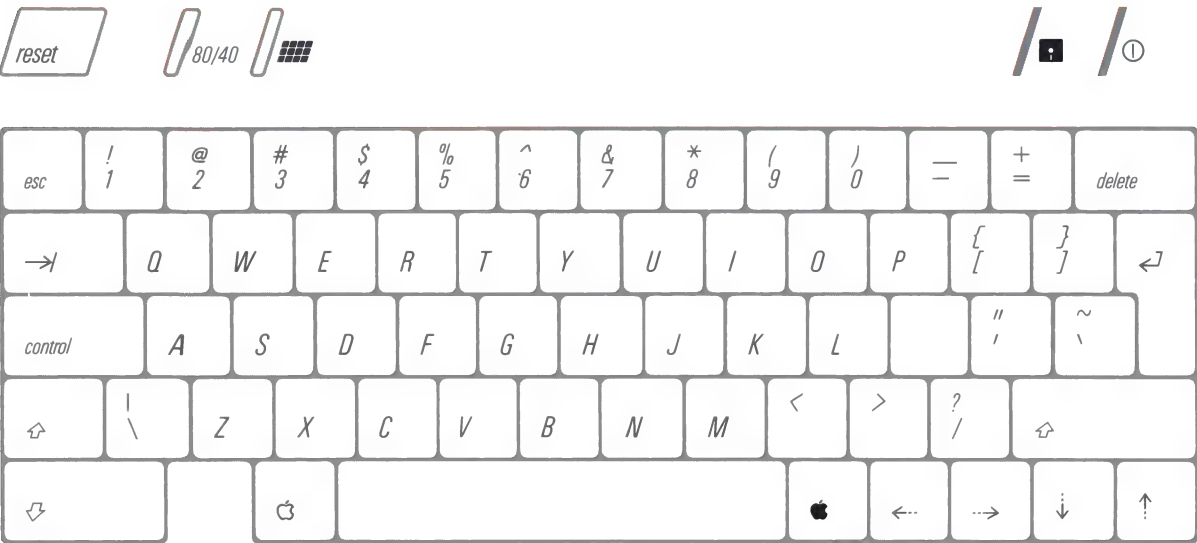
Figure G-2. USA Simplified or Dvorak Keyboard (Keyboard Switch Down)



G.1.3 ISO Layout of USA Keyboard

Figure G-3 shows the layout of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.

Figure G-3. ISO Version of USA Standard Keyboard (Keyboard Switch Up)



G.1.4 English Keyboard

With the keyboard switch up, the English model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the English model keyboard layout is as shown in Figure G-4. The change in ASCII code production (from that in Table G-1) is shown in Table G-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the cross-hatch symbol (#) on the shifted 3-key.

Figure G-4. English Keyboard (Keyboard Switch Down)

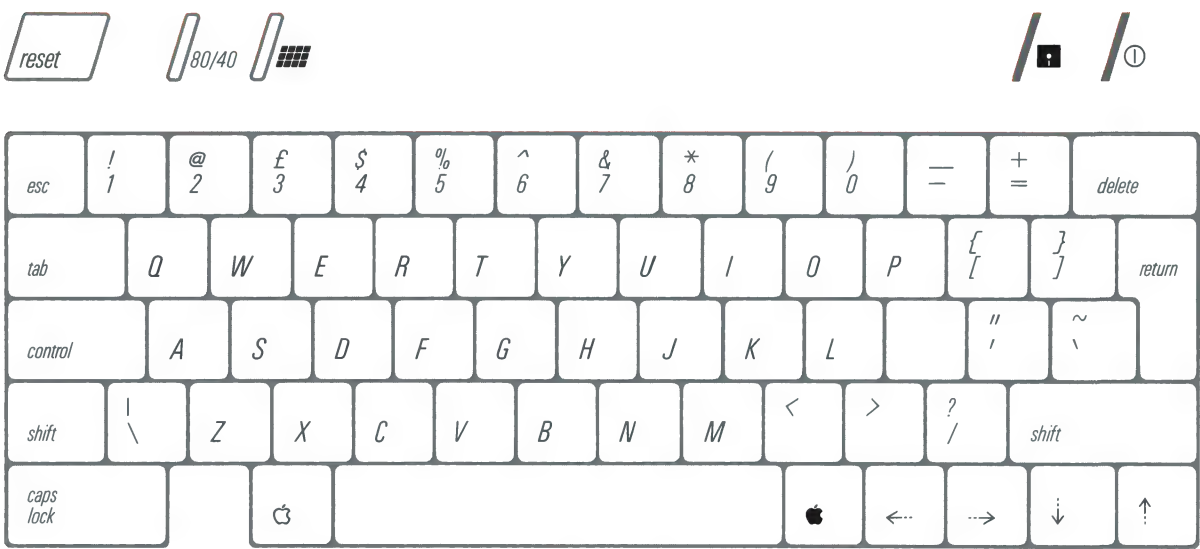


Table G-2. English Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
3 £	33	3	33	3	23	£	23	#

G.1.5 French Keyboard

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Note that on the French keyboard, **[CAPS LOCK]** shifts to the upper characters on all keys. With **[CAPS LOCK]** on, **[SHIFT]** “unshifts” to the lower character on any key pressed with it.

Figure G-5. French Keyboard (Keyboard Switch Down)

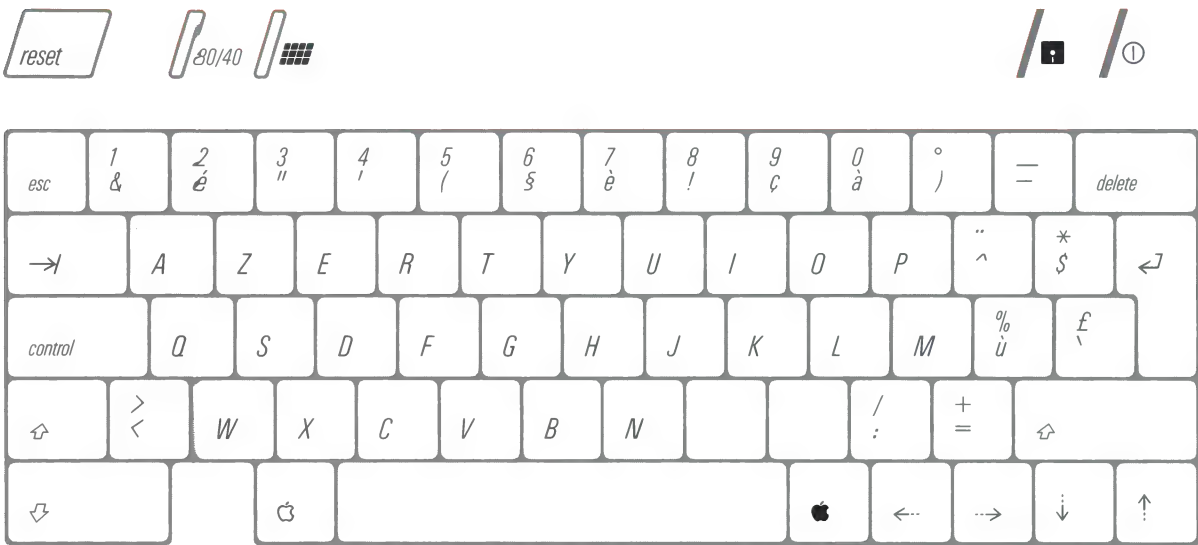


Table G-3. French Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
é 2	7B	é	7B	é	32	2	32	2
" 3	22	"	22		33	3	33	3
' 4	27		27		34	4	34	4
(5	28	(28	(35	5	35	5
§ 6	5D	§	1D	GS	36	6	1D	GS
è 7	7D	è	7D	è	37	7	37	7
! 8	21	!	21	!	38	8	38	8
ç 9	5C	ç	1C	FS	39	9	1C	FS
à 0	40	à	00	NUL	30	0	00	NUL
) °	29)	1B	ESC	5B	°	1B	ESC
^ ~	5E	^	1E	RS	7E		1E	RS
\$ *	24	\$	24	\$	2A	*	2A	*
ù %	7C	ù	7C	ù	25	%	25	%
ˆ £	60		60		23	£	23	£
, ?	2C		2C		3F	?	3F	?
	3B		3B		2E		2E	
: /	3A		3A		2F	/	2F	/

G.1.6 Canadian Keyboard

With the keyboard switch up, the Canadian model of the Apple IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Canadian model keyboard layout is as shown in Figure G-6. The changes in ASCII code production (from that in Table G-1) are shown in Table G-4.

Figure G-6. Canadian Keyboard (Keyboard Switch Down)

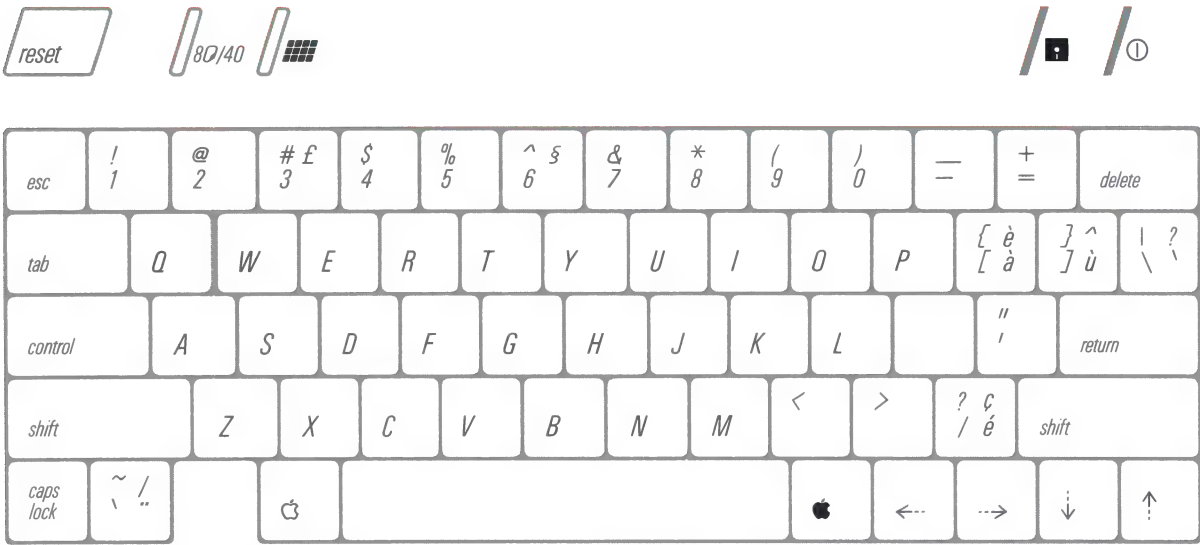


Table G-4. Canadian Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2 °	32	2	00	NUL	5B		00	NUL
3 £	33	3	33	3	23	£	23	£
6 §	36	6	RS	1E	5D	ß	RS	1E
à è	40	à	7F	DEL	7D	è	7F	DEL
ù ^	7C	ù	7C	ù	5E	^	5E	^
` ?	60		ESC	1B	3F	?	1D	GS
é ç	7B	é	1C	FS	5C	ç	1C	FS
" /	7E	"	7E	"	2F	/	2F	/

G.1.7 German Keyboard

With the keyboard switch up, the German model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the German model keyboard layout is as shown in Figure G-7. The change in ASCII code production (from that in Table G-1) is shown in Table G-5.

Figure G-7. German Keyboard (Keyboard Switch Down)

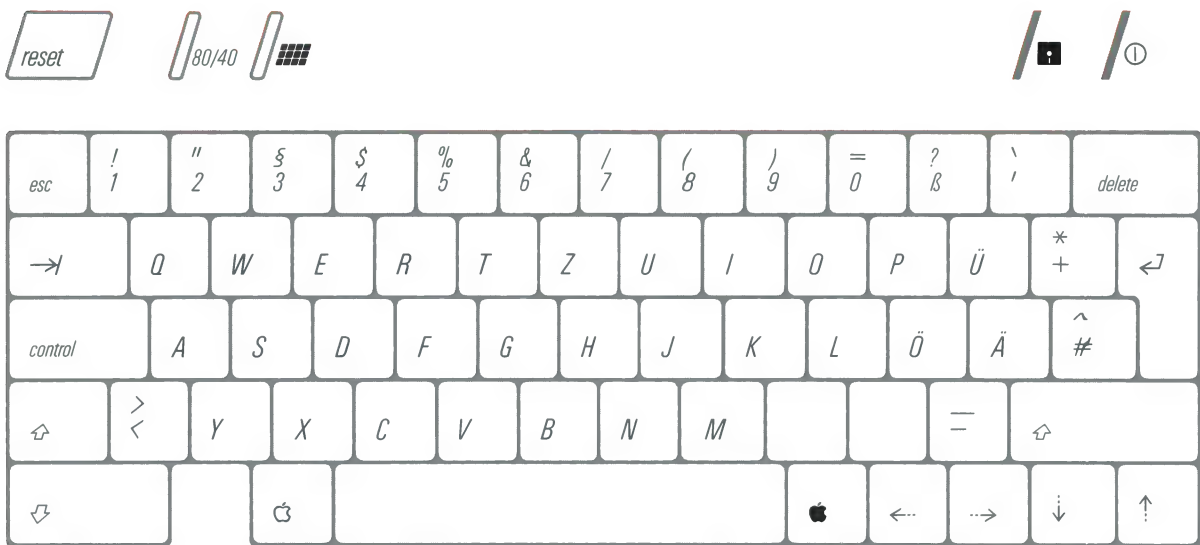


Table G-5. German Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2 "	32	2	32	2	22		22	
3 §	33	3	00	NUL	40	§	00	NUL
6 &	36	6	36	6	26	&	26	&
7 /	37	7	37	7	2F	/	2F	/
8 (38	8	38	8	28	(28	(
9)	39	9	39	9	29)	29)
0 =	30	0	30	0	3D	=	3D	=
ß ?	7E	ß	7E	ß	3F	?	3F	?
Ü	7D	Ü	1D	GS	5D	Ü	1D	GS
+ *	2B	+	2B	+	2A	*	2A	*
Ö	7C	Ö	1C	FS	5C	Ö	1C	FS
Ä	7B	Ä	1B	ESC	5B	Ä	1B	ESC
# ^	23	#	1E	RS	5E	^	1E	RS
< >	3C	<	3C	<	3E	>	3E	>
	2C		2C		3B		3B	
	2E		2E		3A		3A	

G.1.8 Italian Keyboard

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure G-8. The change in ASCII code production (from that in Table G-1) is shown in Table G-6.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters indicated with the circled numbers 0, 2–5, and 7–11 in Table G-8.

Figure G-8. Italian Keyboard (Keyboard Switch Down)

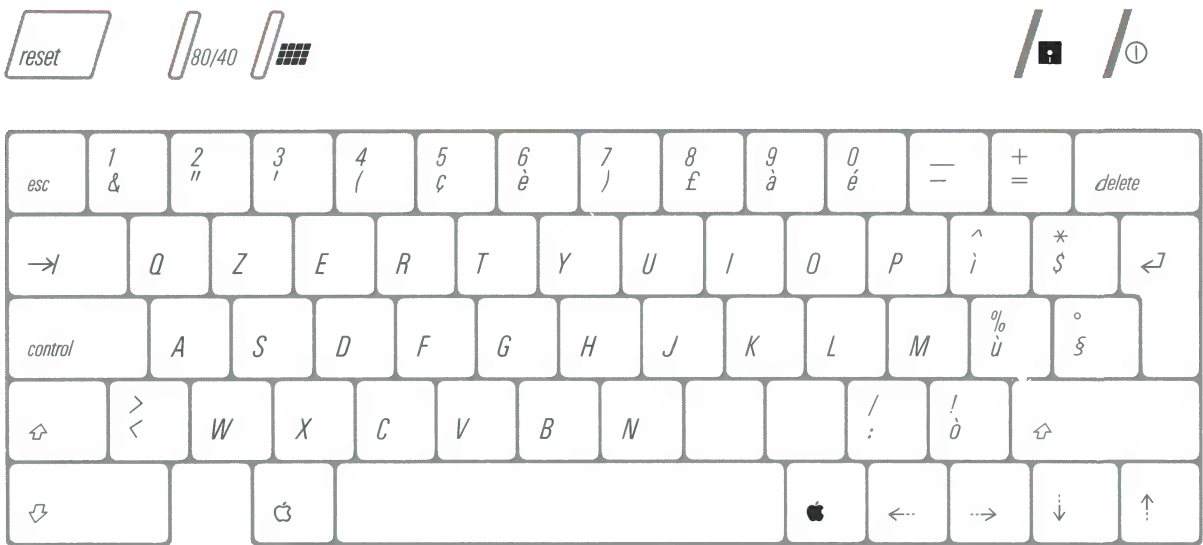


Table G-6. Italian Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
" 2	22	"	22	"	32	2	32	2
' 3	27		27		33	3	33	3
(4	28	(28	(34	4	34	4
ç 5	5C	ç	1C	FS	35	5	1C	FS
è 6	7D	è	7D	è	36	6	36	6
) 7	29)	29)	37	7	37	7
£ 8	23	£	23	£	38	8	38	8
à 9	7B	à	7B	à	39	9	39	9
é 0	5D	é	1D	GS	30	0	1D	GS
ì ^	7E	ì	1E	RS	5E	^	1E	RS
\$ *	24	\$	24	\$	2A	*	2A	*
ù %	60	ù	60	ù	25	%	25	%
§ °	40	§	00	NUL	5B	°	1B	ESC
< >	3C	<	3C	<	3E	>	3E	>
, ?	2C		2C		3F	?	3F	?
	3B		3B		2E		2E	
: /	3A		3A		2F	/	2F	/
ò !	7C	ò	7C	ò	21	!	21	!

G.1.9 Western Spanish Keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure G-9. The change in ASCII code production (from that in Table G-1) is shown in Table G-7.

Figure G-9. Western Spanish Keyboard (Keyboard Switch Down)

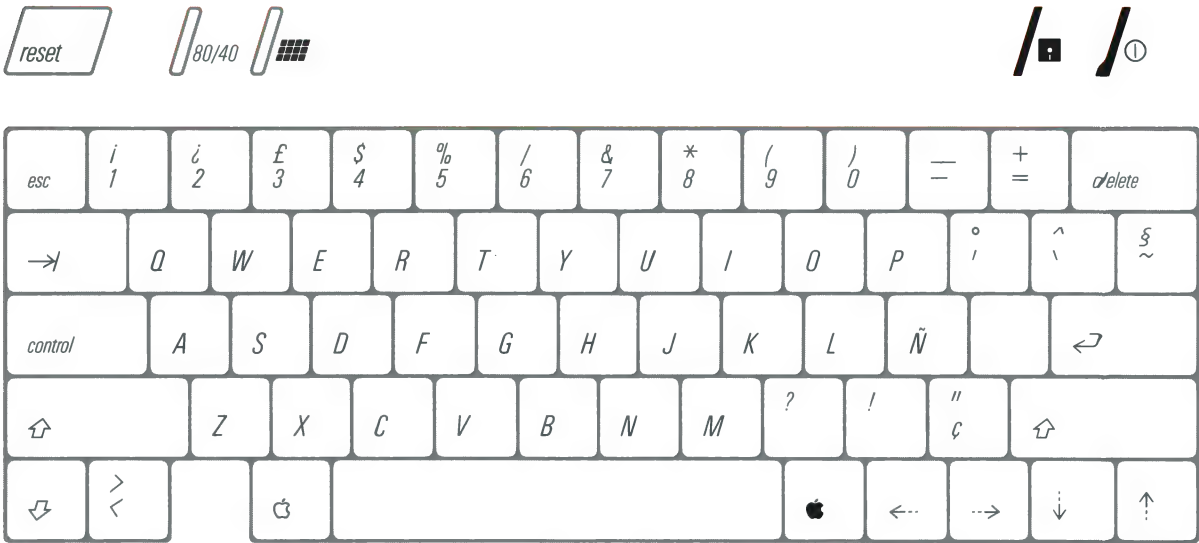


Table G-7. Western Spanish Keyboard Code Differences From Table G-1
Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Key	Key Alone		+ CONTROL		+ SHIFT		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
1 ¡	31	1	31	1	5B	¡	5B	¡
2 ¨	32	2	32	2	5D	¨	5D	¨
3 £	33	3	33	3	23	£	23	
6 /	36	6	36	6	2F	/	2F	/
' °	27	'	27	'	7B	°	7B	°
	60		00	NUL	5E		00	NUL
~ \$	7E	~	7F	DEL	40	\$	7F	DEL
Ñ	7C	ñ	1C	FS	5C	Ñ	1C	FS
, ?	2C		2C		3F	?	3F	?
. !	2E		2E		21	!	21	!
ç "	7D	ç	1D	GS	22	"	1D	GS
< >	3C	<	1E	RS	3E	>	1E	RS

G.2 ASCII Character Sets

Table G-8 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc uses, as well as the decimal and hexadecimal equivalents. Where there are differences between character sets, an asterisked number in the main table refers to a column in the following part of the table.

Table G-8. ASCII Code Equivalents

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
NUL	00	00	SP	32	20	2 *	64	40	7 *	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	0 *	35	23	C	67	43	c	99	63
EOT	04	04	1 *	36	24	D	68	44	d	100	64
ENQ	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67
BS	08	08	(40	28	H	72	48	h	104	68
HT	09	09)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C		44	2C	L	76	4C	l	108	6C
CR	13	0D		45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A		58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	3 *	91	5B	8 *	123	7B
FS	28	1C	<	60	3C	4 *	92	5C	9 *	124	7C
GS	29	1D	=	61	3D	5 *	93	5D	10 *	125	7D
RS	30	1E	>	62	3E	6 *	94	5E	11 *	126	7E
US	31	1F	?	63	3F	—	95	5F	DEL	127	7F

Table G-8—continued. ASCII Code Equivalents
Note: These characters correspond to those followed by an asterisk in the preceding part of the table.

*	0	1	2	3	4	5	6	7	8	9	10	11
Hexadecimal	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
English (USA)	#	\$	@	[\]			{		}	~
English (UK)	£	\$	@	[\]			{		}	
German	#	\$	§	Ä	Ö	Ü			ä	ö	ü	ß
French	£	\$	à		ç	§			é	ù	è	
Italian	£	\$	§		ç	é		ù	à	ò	è	ì
Spanish	£	\$	§	í	Ñ	¿	^		°	ñ	ç	~

G.3 Certification

In the countries where it is applicable, the following product safety certification supplements the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

G.3.1 Product Safety

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

G.3.2 Important Safety Instructions

This equipment is intended to be electrically grounded. This product is equipped with a plug having a third (grounding) pin. This plug will fit only into a grounding-type alternating current outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

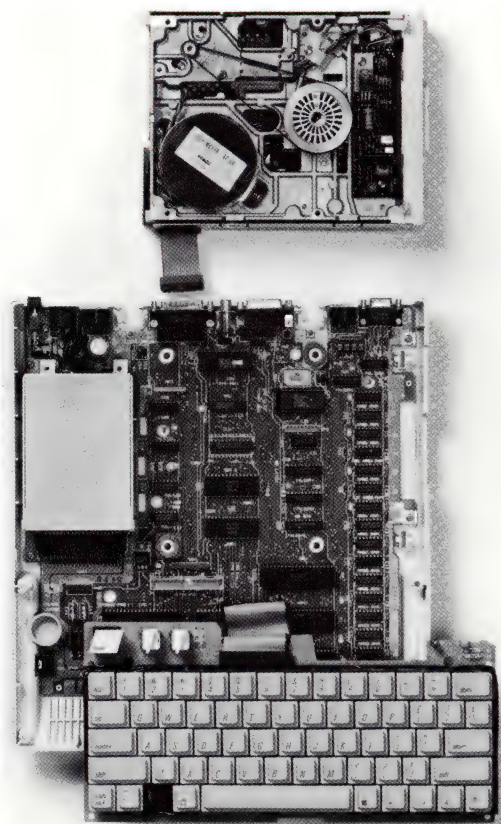
Do not defeat the purpose of the grounding-type plug.

G.4 Power Supply Specifications

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50-Hz alternating current are shown in Table G-8.

Table G-8. 50-Hz Power Supply Specifications

Line voltage:	199 to 255 VAC, 50 Hz
Maximum input power consumption:	25 W
Supply voltage:	+15 VDC (nominal)
Supply current:	1.2 A (nominal)



This appendix briefly discusses bits and bytes and what they can represent, and peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

H.1 Bits and Bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C02:

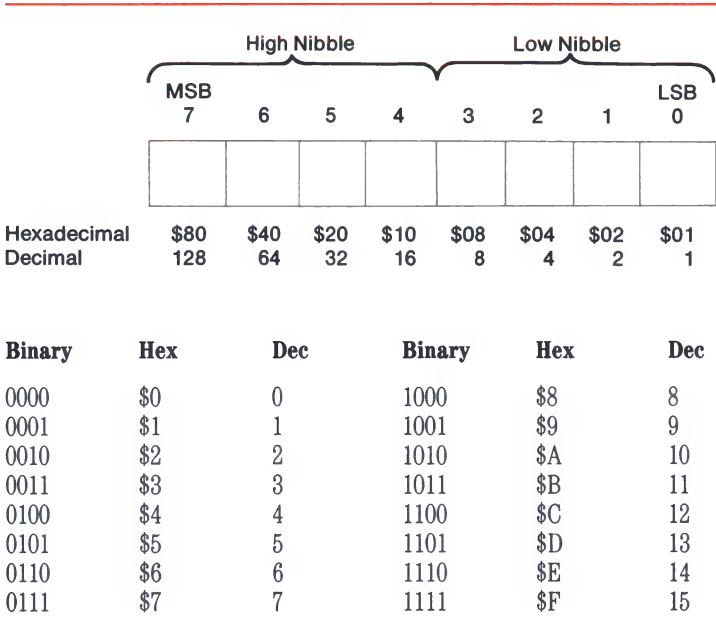
- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc are listed in Table H-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only 10 of the 16 digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).
- One byte can represent any of 16×16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc contains one 8-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables H-5 through H-8 list some of the ways bytes are commonly interpreted.
- Two bytes make a word. The 16 bits of a word can represent any one of 256×256 , or 65536, different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

Table H-1. What a Bit Can Represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

* Sometimes ambiguously termed *reset*.

Figure H-1. Bits, Nibbles, and Bytes



H.2 Hexadecimal and Decimal

Use Table H-2 for conversion of hexadecimal and decimal numbers.

Table H-2. Hexadecimal/Decimal Conversion

Digit	\$x000	\$0x00	\$00x0	\$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

Examples:

\$3C = ?	\$FD47 = ?
\$30 = 48	\$F000 = 61440
\$0C = 12	\$D00 = 3328
-----	\$ 40 = 64
	\$ 7 = 7
\$3C = 60	-----
	\$FD47 = 64839

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have 0 left. Add up the hexadecimal numbers.

Example:

16215 = \$?

16215 - 12288 = 3927

3927 - 3840 = 87

87 - 80 = 7

7

12288 = \$7000

3840 = \$F00

80 = \$50

7 = \$7

16215 = \$7F57

H.3 Hexadecimal and Negative Decimal

If a number is larger than decimal 32767, Applesoft BASIC allows and Integer BASIC requires you to use the negative-decimal equivalent of the number. Table H-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

Table H-3. Hexadecimal to Negative Decimal Conversion

Digit	\$x000	\$0x00	\$00x0	\$000x
F	0	0	0	—1
E	—4096	—256	—16	—2
D	—8192	—512	—32	—3
C	—12288	—768	—48	—4
B	—16384	—1024	—64	—5
A	—20480	—1280	—80	—6
9	—24576	—1536	—96	—7
8	—28672	—1792	—112	—8
7		—2048	—128	—9
6		—2304	—144	—10
5		—2560	—160	—11
4		—2816	—176	—12
3		—3072	—192	—13
2		—3328	—208	—14
1		—3584	—224	—15
0		—3840	—240	—16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0s included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative-decimal number.

Example:

```
$C010 = - ?

$C000:  -12288
$ 000:  - 3840
$  10:  -  224
$   0:  -   16
-----
$C010   -16368
```

To convert a negative-decimal number directly to a positive-decimal number, add it to 65536. (This addition ends up looking like subtraction.)

Example:

```
-151 = + ?

65536 + (-151) = 65536 - 151 = 65385
```

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive-decimal number, then use Table H-2.

H.4 Peripheral Identification Numbers

Many Apple products now use peripheral identification numbers (called PIN numbers) as shorthand to designate serial device characteristics. The Apple II series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table H-4 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

Example: 252/1111 means:

Communication mode	No parity
8 data bits, 1 stop bit	Do not echo output to display
300 baud (bits per second)	No line feed after carriage return
	Do not generate carriage returns

Table H-4. PIN Numbers

	x	x	x	/	x	x	x	x
1 = Printer mode 2 = Communication mode*								
1 = 6 data bits, 1 stop bit 2 = 6 data bits, 2 stop bits 3 = 7 data bits, 1 stop bit 4 = 7 data bits, 2 stop bits 5 = 8 data bits, 1 stop bit 6 = 8 data bits, 2 stop bits								
1 = 110 bits per second 2 = 300 bits per second 3 = 1200 bits per second 4 = 2400 bits per second 5 = 4800 bits per second 6 = 9600 bits per second 7 = 19200 bits per second								
1 = No parity 2 = Even parity (total on = even) 3 = Odd parity (total on = odd) 4 = MARK parity (parity bit = 1) 5 = SPACE parity (parity bit = 0)								
1 = Do not echo output on screen 2 = Echo output on screen								
1 = Do not generate LF after CR 2 = Generate LF after CR								
1 = Do not generate CR* 2 = Generate CR after 40 characters 3 = Generate CR after 72 characters 4 = Generate CR after 80 characters 5 = Generate CR after 132 characters								

* If you select communication mode, then seventh digit must equal 1. This value is supplied automatically when you use the UUD.

H.5 Eight-Bit Code Conversions

Tables H-5 through H-8 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above \$7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *H*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity *H*, 01001000 for an even-parity *H*.
- The *ASCII Char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to Type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Pri* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

Note that the values \$40 through \$5F (and \$C0 through \$DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so (Section 5.2.2), or if the firmware is bypassed.

Note: The primary and alternate displayed character sets in Tables H-5 through H-8 are the result of firmware mapping. The character generator ROM actually contains only one character set. The firmware mapping procedure is described in Section 3.3.6.

Table H-5. Control Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0000000	0	\$00	NUL	Blank (null)	CONTROL-@	@	@
0000001	1	\$01	SOH	Start of Header	CONTROL-A	A	A
0000010	2	\$02	STX	Start of Text	CONTROL-B	B	B
0000011	3	\$03	ETX	End of Text	CONTROL-C	C	C
0000100	4	\$04	EOT	End of Transm.	CONTROL-D	D	D
0000101	5	\$05	ENQ	Enquiry	CONTROL-E	E	E
0000110	6	\$06	ACK	Acknowledge	CONTROL-F	F	F
0000111	7	\$07	BEL	Bell	CONTROL-G	G	G
0001000	8	\$08	BS	Backspace	CONTROL-H or ←	H	H
0001001	9	\$09	HT	Horizontal Tab	CONTROL-I or TAB	I	I
0001010	10	\$0A	LF	Line Feed	CONTROL-J or ↓	J	J
0001011	11	\$0B	VT	Vertical Tab	CONTROL-K or ↑	K	K
0001100	12	\$0C	FF	Form Feed	CONTROL-L	L	L
0001101	13	\$0D	CR	Carriage Return	CONTROL-M or RETURN	M	M
0001110	14	\$0E	SO	Shift Out	CONTROL-N	N	N
0001111	15	\$0F	SI	Shift In	CONTROL-O	O	O
0010000	16	\$10	DLE	Data Link Escape	CONTROL-P	P	P
0010001	17	\$11	DC1	Device Control 1	CONTROL-Q	Q	Q
0010010	18	\$12	DC2	Device Control 2	CONTROL-R	R	R
0010011	19	\$13	DC3	Device Control 3	CONTROL-S	S	S
0010100	20	\$14	DC4	Device Control 4	CONTROL-T	T	T
0010101	21	\$15	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
0010110	22	\$16	SYN	Synchronization	CONTROL-V	V	V
0010111	23	\$17	ETB	End of Text Blk.	CONTROL-W	W	W
0011000	24	\$18	CAN	Cancel	CONTROL-X	X	X
0011001	25	\$19	EM	End of Medium	CONTROL-Y	Y	Y
0011010	26	\$1A	SUB	Substitute	CONTROL-Z	Z	Z
0011011	27	\$1B	ESC	Escape	CONTROL-[or ESC	[[
0011100	28	\$1C	FS	File Separator	CONTROL-\	\	\
0011101	29	\$1D	GS	Group Separator	CONTROL-]]]
0011110	30	\$1E	RS	Record Separator	CONTROL-^	^	^
0011111	31	\$1F	US	Unit Separator	CONTROL-`	—	—

Table H-6. Special Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0100000	32	\$20	SP	Space	SPACE bar		
0100001	33	\$21	!			!	!
0100010	34	\$22	"			"	"
0100011	35	\$23	#			#	#
0100100	36	\$24	\$			\$	\$
0100101	37	\$25	%			%	%
0100110	38	\$26	&			&	&
0100111	39	\$27	'	Closing quote		'	'
0101000	40	\$28	(((
0101001	41	\$29)))
0101010	42	\$2A	*			*	*
0101011	43	\$2B	+			+	+
0101100	44	\$2C	,	Comma		,	,
0101101	45	\$2D	-	Hyphen		-	-
0101110	46	\$2E	.	Period		.	.
0101111	47	\$2F	/			/	/
0110000	48	\$30	0			0	0
0110001	49	\$31	1			1	1
0110010	50	\$32	2			2	2
0110011	51	\$33	3			3	3
0110100	52	\$34	4			4	4
0110101	53	\$35	5			5	5
0110110	54	\$36	6			6	6
0110111	55	\$37	7			7	7
0111000	56	\$38	8			8	8
0111001	57	\$39	9			9	9
0111010	58	\$3A	:			:	:
0111011	59	\$3B	;			;	;
0111100	60	\$3C	<			<	<
0111101	61	\$3D	=			=	=
0111110	62	\$3E	>			>	>
0111111	63	\$3F	?			?	?

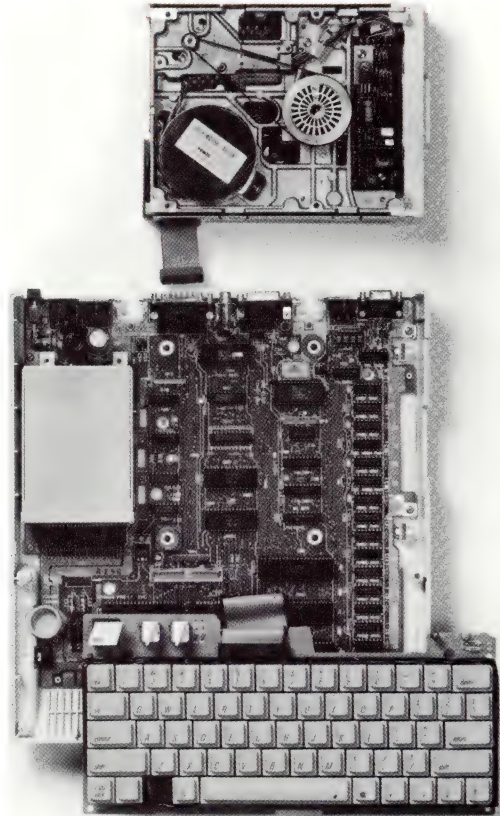
Table H-7. Uppercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt*
1000000	64	\$40	@			@	⌘
1000001	65	\$41	A			A	⌘
1000010	66	\$42	B			B	⌘
1000011	67	\$43	C			C	⌘
1000100	68	\$44	D			D	⌘
1000101	69	\$45	E			E	⌘
1000110	70	\$46	F			F	⌘
1000111	71	\$47	G			G	⌘
1001000	72	\$48	H			H	⌘
1001001	73	\$49	I			I	⌘
1001010	74	\$4A	J			J	⌘
1001011	75	\$4B	K			K	⌘
1001100	76	\$4C	L			L	⌘
1001101	77	\$4D	M			M	⌘
1001110	78	\$4E	N			N	⌘
1001111	79	\$4F	O			O	⌘
1010000	80	\$50	P			P	⌘
1010001	81	\$51	Q			Q	⌘
1010010	82	\$52	R			R	⌘
1010011	83	\$53	S			S	⌘
1010100	84	\$54	T			T	⌘
1010101	85	\$55	U			U	⌘
1010110	86	\$56	V			V	⌘
1010111	87	\$57	W			W	⌘
1011000	88	\$58	X			X	⌘
1011001	89	\$59	Y			Y	⌘
1011010	90	\$5A	Z			Z	⌘
1011011	91	\$5B	[Opening bracket		/	⌘
1011100	92	\$5C	\	Reverse slant		\	⌘
1011101	93	\$5D]	Closing bracket		/	⌘
1011110	94	\$5E	^	Caret		^	⌘
1011111	95	\$5F	_	Underline		_	⌘

* If the high bit is set, the MouseText characters are replaced with their equivalent in the primary character set with that value.

Table H-8. Lowercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1100000	96	\$60		Opening quote			
1100001	97	\$61	a			!	a
1100010	98	\$62	b			"	b
1100011	99	\$63	c			#	c
1100100	100	\$64	d			\$	d
1100101	101	\$65	e			%	e
1100110	102	\$66	f			&	f
1100111	103	\$67	g			'	g
1101000	104	\$68	h			(h
1101001	105	\$69	i)	i
1101010	106	\$6A	j			*	j
1101011	107	\$6B	k			+	k
1101100	108	\$6C	l				l
1101101	109	\$6D	m				m
1101110	110	\$6E	n			.	n
1101111	111	\$6F	o			/	o
1110000	112	\$70	p			0	p
1110001	113	\$71	q			1	q
1110010	114	\$72	r			2	r
1110011	115	\$73	s			3	s
1110100	116	\$74	t			4	t
1110101	117	\$75	u			5	u
1110110	118	\$76	v			6	v
1110111	119	\$77	w			7	w
1111000	120	\$78	x			8	x
1111001	121	\$79	y			9	y
1111010	122	\$7A	z				z
1111011	123	\$7B	{	Opening brace		;	{
1111100	124	\$7C		Vertical line		<	
1111101	125	\$7D	}	Closing brace		=	}
1111110	126	\$7E	~	Overline (tilde)		>	~
1111111	127	\$7F	DEL	Delete/rubout		?	DEL



Appendix I is a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the version of the Apple IIc that supports UniDisk 3.5.

If you find that you absolutely must develop software or hardware for an original Apple IIc that will not be upgraded in the future, you can get a listing of the old source code by filling out and mailing the order form in the back of this manual.

[illegible]

0844	73	•	MACSTAT	EQ0	844
0845	75	ACC	EQ0	845	
	76	•			
0846	77	XREG	EQ0	846	
0847	78	REG	EQ0	847	
0848	79	SPRINT	EQ0	848	
0849	80	SPMT	EQ0	849	
084E	81	RNDH	EQ0	84E	
084F	82	RNDH	EQ0	84F	
	83	•			
085	84	•	Value equates		
086	85	•			
0886	86	GOODF8	EQ0	886	
0895	87	PICK	EQ0	895	
089B	88	ESC	EQ0	89B	
	89	•			
089C	90	•	Characters read by		
089D	91	•	1h. terminated by a		
089E	92	•	1h. terminated by a		
089F	93	IN	EQ0	8200	
	94	•			
089	95	•	Page 3 vectors		
	96	•			
089F	97	BRKV	EQ0	893F	
08E2	98	SOFTV	EQ0	893E	
08F4	99	PAIREDUP	EQ0	893F4	
08F5	100	ANPEV	EQ0	893F5	
08F8	101	USRADR	EQ0	893F8	
08F9	102	NMI	EQ0	893F9	
08FB	103	LDI	EQ0	893FB	
08FC	104	LINE	EQ0	893FC	
08FE	105	MSLDT	EQ0	893FE	
08FB	106	•			
	107	•	HARDWARE EQUATES		
0800	108	ADDR	EQ0	8C000	
0801	109	KEN	EQ0	8C001	
0808	110	CLRR8COL	EQ0	8C008	
0801	112	ST8BCOL	EQ0	8C001	
0802	113	RDMAINRAM	EQ0	8C002	
0803	114	ROCDRAM	EQ0	8C003	
0804	115	RDMAINRAM	EQ0	8C004	
0805	116	WCRADRAM	EQ0	8C005	
0808	117	SETSDZP	EQ0	8C008	
0809	118	SETALZP	EQ0	8C009	
080C	119	LR8BVID	EQ0	8C00C	
080D	120	ST8BVID	EQ0	8C00D	
080E	121	ST8BVID	EQ0	8C00E	
080F	122	SETALZCHGR	EQ0	8C00F	
0810	123	RDLSTRB	EQ0	8C010	
0811	124	KDCLDNK2	EQ0	8C011	
0812	125	RDLRCAR	EQ0	8C012	
0813	126	RDRAHND	EQ0	8C013	
0814	127	RDRAWMT	EQ0	8C014	
0815	128	RDRAWMT	EQ0	8C015	
0816	129	RDBMCL	EQ0	8C016	
0817	130	RDBMCL	EQ0	8C017	
0819	131	RDTEXT	EQ0	8C019	
081A	132	RDMIX	EQ0	8C01A	
081B	132	RDMIX	EQ0	8C01B	
081C	133	RDPAGE2	EQ0	8C01C	
081D	134	RDRHRES	EQ0	8C01D	
081E	135	RDRHRES	EQ0	8C01E	
081F	136	RDHRES	EQ0	8C01F	
0820	137	RDHRES	EQ0	8C020	
0828	138	RDHRES	EQ0	8C028	
0830	138	SPKR	EQ0	8C030	
0850	139	TATCLR	EQ0	8C050	
0851	140	TATSET	EQ0	8C051	
0852	141	TATSET	EQ0	8C052	
0853	142	MTSET	EQ0	8C053	
0854	143	TATPAGE1	EQ0	8C054	

```

;Machine state after BRK
;Acc after BRK

;X reg after break
;Y reg after break
;SP reg after break
;random counter low
;random counter high

;value of //e, lolly ID byte
;CONTROL-U character
;what ESC generates

TLN are placed in
;carriage return.

;input buffer for GETLN

;vectors here after break
;vectors here after break
;THIS MUST = EOR #485 OF SOFTEV+1
;APPLESOFT & EXIT VECTOR
;APPLESOFT USR function vector
;NMI vector
;maskable interrupt vector
;maskable interrupt screen
;lower of $C8 space vector

;for IMV, PPV vector
;?27 if 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806
```


C100:
C100:

```

3 .....
4 .....
5 Apple Lolly communications driver
6 .....
7 Bc
8 .....
9 Rich Williams
10 .....
11 November 5 - j.r.huston
12 .....
13 .....
14 .....
15 Command codes
16 .....
17 *AnnB: Set baud rate to nn
18 *AnnC: Set video rate to nn
19 *AI: Enable video echo
20 *AK: Disable CRLF
21 *AL: Enable CRLF
22 *AN: Disable video echo & set printer
23 *AnnD: Get parity bits to nn
24 *AnnE: Get parity bits to nn
25 *AR: Reset the ACIA, IN# PR#B
26 *AS: Enter terminal mode
27 *AT: Enter terminal mode
28 *AZ: Set control commands
29 *AZ: Set control commands
30 *AZ: Set control commands
31 *AnnCR:Set printer width (CR = carriage
32 .....
33 * New commands added in rev 1 E = enable/disable
34 *AC E/D Column overflow
35 *AL E/D Linefeed same as L & K
36 *AM E/D Mask incoming linefeeds
37 *AX E/D Xon Xoff handshaking
38 *AF E/D Find keyboard
39 .....
40 .....
41 .....
42 .....
43 .....
44 .....
45 .....
46 .....
47 .....
48 .....
49 .....
50 .....
51 .....
52 .....
53 .....
54 .....
55 .....
56 .....
57 .....
58 .....
59 .....
60 .....
61 .....
62 .....
63 .....
64 .....
65 .....
66 .....
67 .....
68 .....
69 .....
70 .....
71 .....
72 .....
73 .....

```


86 COMM	Communications port routine	26-JUL-85	PAGE 11	86 COMM	Communications port routine	26-JUL-85	PAGE 12
C280:2C 89 C1	3 bit sersts			C276:FA	74 pix		
C283:78 14 C219	4 bva			C277:68	75 ris		
C285:38 sec	5 sin			C278:18	76 goremte c1c		
C287:18 d1b	6 d1b			C27C:28 A3 C7 C27C	78 gterm jar		
C289:58 8C	7 sout			C27D:28 4C CC	79 term1 jar		
C289:58 8E C219	8 bvc			C27E:28 4C CC	80 serin jar		
C289:91	9			C280:28 D9 C7 C28E	81 sinokbd jar		
C289:91	10			C283:88 89	82 bva		
C290:11	11 d1b			C284:29 18	83 bva		
C290:11	12 d1b			C285:29 18	84 bva		
C290:11	13 d1b			C286:29 18	85 bva		
C290:11	14 d1b			C287:29 18	86 bva		
C290:11	15 d1b			C288:29 18	87 bva		
C290:11	16 d1b			C289:29 18	88 bva		
C290:11	17 d1b			C290:29 18	89 bva		
C290:11	18 * Pascal support stuff			C291:29 18	90 bva		
C291:88 BB C19E	20 p2init bra p1init			C292:29 18	91 bva		
C291:88 93 C188	21 p2read bra p1read			C293:29 18	92 bva		
C291:88 95 C188	22 p2write bra p1write			C294:29 18	93 bva		
C291:88 97 C188	23 p2status bra p1status			C295:29 18	94 bva		
C291:88 99 C188	24 p2init bra p1init			C296:29 18	95 bva		
C291:88 9B C188	25 p2read bra p1read			C297:29 18	96 bva		
C291:88 9D C188	26 p2write bra p1write			C298:29 18	97 bva		
C291:88 9F C188	27 p2status bra p1status			C299:29 18	98 bva		
C291:88 A1 C188	28 p2init bra p1init			C300:29 18	99 bva		
C291:88 A3 C188	29 p2read bra p1read			C301:29 18	100 bva		
C291:88 A5 C188	30 p2write bra p1write			C302:29 18	101 bva		
C291:88 A7 C188	31 p2status bra p1status			C303:29 18	102 bva		
C291:88 A9 C188	32 p2init bra p1init			C304:29 18	103 bva		
C291:88 AB C188	33 p2read bra p1read			C305:29 18	104 bva		
C291:88 AD C188	34 p2write bra p1write			C306:29 18	105 bva		
C291:88 AF C188	35 p2status bra p1status			C307:29 18	106 bva		
C291:88 B1 C188	36 p2init bra p1init			C308:29 18	107 bva		
C291:88 B3 C188	37 p2read bra p1read			C309:29 18	108 bva		
C291:88 B5 C188	38 p2write bra p1write			C310:29 18	109 bva		
C291:88 B7 C188	39 p2status bra p1status			C311:29 18	110 bva		
C291:88 B9 C188	40 p2init bra p1init			C312:29 18	111 bva		
C291:88 BB C188	41 p2read bra p1read			C313:29 18	112 bva		
C291:88 BD C188	42 p2write bra p1write			C314:29 18	113 bva		
C291:88 BF C188	43 p2status bra p1status			C315:29 18	114 bva		
C291:88 C1 C188	44 p2init bra p1init			C316:29 18	115 bva		
C291:88 C3 C188	45 p2read bra p1read			C317:29 18	116 bva		
C291:88 C5 C188	46 p2write bra p1write			C318:29 18	117 bva		
C291:88 C7 C188	47 p2status bra p1status			C319:29 18	118 bva		
C291:88 C9 C188	48 p2init bra p1init			C320:29 18	119 bva		
C291:88 CB C188	49 p2read bra p1read			C321:29 18	120 bva		
C291:88 CD C188	50 p2write bra p1write			C322:29 18	121 bva		
C291:88 CF C188	51 p2status bra p1status			C323:29 18	122 bva		
C291:88 D1 C188	52 p2init bra p1init			C324:29 18	123 bva		
C291:88 D3 C188	53 p2read bra p1read			C325:29 18	124 bva		
C291:88 D5 C188	54 p2write bra p1write			C326:29 18	125 bva		
C291:88 D7 C188	55 p2status bra p1status			C327:29 18	126 bva		
C291:88 D9 C188	56 p2init bra p1init			C328:29 18	127 bva		
C291:88 DB C188	57 p2read bra p1read			C329:29 18	128 bva		
C291:88 DD C188	58 p2write bra p1write			C330:29 18	129 bva		
C291:88 DF C188	59 p2status bra p1status			C331:29 18	130 bva		
C291:88 E1 C188	60 p2init bra p1init			C332:29 18	131 bva		
C291:88 E3 C188	61 p2read bra p1read			C333:29 18	132 bva		
C291:88 E5 C188	62 p2write bra p1write			C334:29 18	133 bva		
C291:88 E7 C188	63 p2status bra p1status			C335:29 18	134 bva		
C291:88 E9 C188	64 p2init bra p1init			C336:29 18	135 bva		
C291:88 EB C188	65 p2read bra p1read			C337:29 18	136 bva		
C291:88 ED C188	66 p2write bra p1write			C338:29 18	137 bva		
C291:88 EF C188	67 p2status bra p1status			C339:29 18	138 bva		
C291:88 F1 C188	68 p2init bra p1init			C340:29 18	139 bva		
C291:88 F3 C188	69 p2read bra p1read			C341:29 18	140 bva		
C291:88 F5 C188	70 p2write bra p1write			C342:29 18	141 bva		
C291:88 F7 C188	71 p2status bra p1status			C343:29 18	142 bva		
C291:88 F9 C188	72 p2init bra p1init			C344:29 18	143 bva		
C291:88 FB C188	73 p2read bra p1read			C345:29 18	144 bva		
C291:88 FD C188	74 p2write bra p1write			C346:29 18	145 bva		
C291:88 FF C188	75 p2status bra p1status			C347:29 18	146 bva		

```

C380: 2 *****
C381: 3 * THIS IS THE $C3XX ROM SPACE:
C382: 4 *****
C383: 5 *****
C384: 6 *****
C385: 7 C3ENTRY PHA :save regs
C386: 8 *****
C387: 9 *****
C388: 10 *****
C389: 11 *****
C390: 12 *****
C391: 13 *****
C392: 14 *****
C393: 15 *****
C394: 16 *****
C395: 17 *****
C396: 18 *****
C397: 19 *****
C398: 20 *****
C399: 21 *****
C400: 22 *****
C401: 23 *****
C402: 24 *****
C403: 25 *****
C404: 26 *****
C405: 27 *****
C406: 28 *****
C407: 29 *****
C408: 30 *****
C409: 31 *****
C410: 32 *****
C411: 33 *****
C412: 34 *****
C413: 35 *****
C414: 36 *****
C415: 37 *****
C416: 38 *****
C417: 39 *****
C418: 40 *****
C419: 41 *****
C420: 42 *****
C421: 43 *****
C422: 44 *****
C423: 45 *****
C424: 46 *****
C425: 47 *****
C426: 48 *****
C427: 49 *****
C428: 50 *****
C429: 51 *****
C430: 52 *****
C431: 53 *****
C432: 54 *****
C433: 55 *****
C434: 56 *****
C435: 57 *****
C436: 58 *****
C437: 59 *****
C438: 60 *****
C439: 61 *****
C440: 62 *****
C441: 63 *****
C442: 64 *****
C443: 65 *****
C444: 66 *****
C445: 67 *****
C446: 68 *****
C447: 69 *****
C448: 70 *****
C449: 71 *****
C450: 72 *****

```

```

C3A4:92 36 STA (CSWL)
C3A5:93 37 INC
C3A6:94 38 INC
C3A7:95 39 INC
C3A8:96 40 INC
C3A9:97 41 INC
C3AA:98 42 INC
C3AB:99 43 INC
C3AC:00 44 INC
C3AD:01 45 INC
C3AE:02 46 INC
C3AF:03 47 INC
C3B0:04 48 INC
C3B1:05 49 INC
C3B2:06 50 INC
C3B3:07 51 INC
C3B4:08 52 INC
C3B5:09 53 INC
C3B6:10 54 INC
C3B7:11 55 INC
C3B8:12 56 INC
C3B9:13 57 INC
C3BA:14 58 INC
C3BB:15 59 INC
C3BC:16 60 INC
C3BD:17 61 INC
C3BE:18 62 INC
C3BF:19 63 INC
C3C0:20 64 INC
C3C1:21 65 INC
C3C2:22 66 INC
C3C3:23 67 INC
C3C4:24 68 INC
C3C5:25 69 INC
C3C6:26 70 INC
C3C7:27 71 INC
C3C8:28 72 INC
C3C9:29 73 INC
C3CA:30 74 INC
C3CB:31 75 INC
C3CC:32 76 INC
C3CD:33 77 INC
C3CE:34 78 INC
C3CF:35 79 INC
C3D0:36 80 INC
C3D1:37 81 INC
C3D2:38 82 INC
C3D3:39 83 INC
C3D4:40 84 INC
C3D5:41 85 INC
C3D6:42 86 INC
C3D7:43 87 INC
C3D8:44 88 INC
C3D9:45 89 INC
C3DA:46 90 INC
C3DB:47 91 INC
C3DC:48 92 INC
C3DD:49 93 INC
C3DE:50 94 INC
C3DF:51 95 INC
C3E0:52 96 INC
C3E1:53 97 INC
C3E2:54 98 INC
C3E3:55 99 INC
C3E4:00 00 INC
C3E5:01 01 INC
C3E6:02 02 INC
C3E7:03 03 INC
C3E8:04 04 INC
C3E9:05 05 INC
C3EA:06 06 INC
C3EB:07 07 INC
C3EC:08 08 INC
C3ED:09 09 INC
C3EE:10 10 INC
C3EF:11 11 INC
C3F0:12 12 INC
C3F1:13 13 INC
C3F2:14 14 INC
C3F3:15 15 INC
C3F4:16 16 INC
C3F5:17 17 INC
C3F6:18 18 INC
C3F7:19 19 INC
C3F8:20 20 INC
C3F9:21 21 INC
C3FA:22 22 INC
C3FB:23 23 INC
C3FC:24 24 INC
C3FD:25 25 INC
C3FE:26 26 INC
C3FF:27 27 INC
C400:28 28 INC
C401:29 29 INC
C402:30 30 INC
C403:31 31 INC
C404:32 32 INC
C405:33 33 INC
C406:34 34 INC
C407:35 35 INC
C408:36 36 INC
C409:37 37 INC
C410:38 38 INC
C411:39 39 INC
C412:40 40 INC
C413:41 41 INC
C414:42 42 INC
C415:43 43 INC
C416:44 44 INC
C417:45 45 INC
C418:46 46 INC
C419:47 47 INC
C420:48 48 INC
C421:49 49 INC
C422:50 50 INC
C423:51 51 INC
C424:52 52 INC
C425:53 53 INC
C426:54 54 INC
C427:55 55 INC
C428:56 56 INC
C429:57 57 INC
C430:58 58 INC
C431:59 59 INC
C432:60 60 INC
C433:61 61 INC
C434:62 62 INC
C435:63 63 INC
C436:64 64 INC
C437:65 65 INC
C438:66 66 INC
C439:67 67 INC
C440:68 68 INC
C441:69 69 INC
C442:70 70 INC
C443:71 71 INC
C444:72 72 INC
C445:73 73 INC
C446:74 74 INC
C447:75 75 INC
C448:76 76 INC
C449:77 77 INC
C450:78 78 INC

```


96

Mouse firmware

```

C50F: 66 xmbasic equ
C50F: 67 pty
C50F: 68 ldy
C50F: 69 ldx
C50F: 70 kwh
C50F: 71 kwh
C50F: 72 kwh
C50F: 73 kwh
C50F: 74 kwh
C50F: 75 kwh
C50F: 76 kwh
C50F: 77 kwh
C50F: 78 kwh
C50F: 79 kwh
C50F: 80 kwh
C50F: 81 kwh
C50F: 82 kwh
C50F: 83 kwh
C50F: 84 kwh
C50F: 85 kwh
C50F: 86 kwh
C50F: 87 kwh
C50F: 88 kwh
C50F: 89 kwh
C50F: 90 kwh
C50F: 91 kwh
C50F: 92 kwh
C50F: 93 kwh
C50F: 94 kwh
C50F: 95 kwh
C50F: 96 kwh
C50F: 97 kwh
C50F: 98 kwh
C50F: 99 kwh
C50F: 100 kwh

```

Disk II boot code

```

C600: 4 *****
C600: 5 *****
C600: 6 *****
C600: 7 *****
C600: 8 *****
C600: 9 *****
C600: 10 *****
C600: 11 *****
C600: 12 *****
C600: 13 *****
C600: 14 *****
C600: 15 *****
C600: 16 *****
C600: 17 *****
C600: 18 *****
C600: 19 *****
C600: 20 *****
C600: 21 *****
C600: 22 *****
C600: 23 *****
C600: 24 *****
C600: 25 *****
C600: 26 *****
C600: 27 *****
C600: 28 *****
C600: 29 *****
C600: 30 *****
C600: 31 *****
C600: 32 *****
C600: 33 *****
C600: 34 *****
C600: 35 *****
C600: 36 *****
C600: 37 *****
C600: 38 *****
C600: 39 *****
C600: 40 *****
C600: 41 *****
C600: 42 *****
C600: 43 *****
C600: 44 *****
C600: 45 *****
C600: 46 *****
C600: 47 *****
C600: 48 *****
C600: 49 *****
C600: 50 *****
C600: 51 *****
C600: 52 *****
C600: 53 *****
C600: 54 *****
C600: 55 *****
C600: 56 *****
C600: 57 *****
C600: 58 *****
C600: 59 *****
C600: 60 *****
C600: 61 *****
C600: 62 *****
C600: 63 *****
C600: 64 *****
C600: 65 *****
C600: 66 *****
C600: 67 *****
C600: 68 *****
C600: 69 *****
C600: 70 *****
C600: 71 *****
C600: 72 *****
C600: 73 *****
C600: 74 *****

```

11 BOOT

```

C600: 4 *****
C600: 5 *****
C600: 6 *****
C600: 7 *****
C600: 8 *****
C600: 9 *****
C600: 10 *****
C600: 11 *****
C600: 12 *****
C600: 13 *****
C600: 14 *****
C600: 15 *****
C600: 16 *****
C600: 17 *****
C600: 18 *****
C600: 19 *****
C600: 20 *****
C600: 21 *****
C600: 22 *****
C600: 23 *****
C600: 24 *****
C600: 25 *****
C600: 26 *****
C600: 27 *****
C600: 28 *****
C600: 29 *****
C600: 30 *****
C600: 31 *****
C600: 32 *****
C600: 33 *****
C600: 34 *****
C600: 35 *****
C600: 36 *****
C600: 37 *****
C600: 38 *****
C600: 39 *****
C600: 40 *****
C600: 41 *****
C600: 42 *****
C600: 43 *****
C600: 44 *****
C600: 45 *****
C600: 46 *****
C600: 47 *****
C600: 48 *****
C600: 49 *****
C600: 50 *****
C600: 51 *****
C600: 52 *****
C600: 53 *****
C600: 54 *****
C600: 55 *****
C600: 56 *****
C600: 57 *****
C600: 58 *****
C600: 59 *****
C600: 60 *****
C600: 61 *****
C600: 62 *****
C600: 63 *****
C600: 64 *****
C600: 65 *****
C600: 66 *****
C600: 67 *****
C600: 68 *****
C600: 69 *****
C600: 70 *****
C600: 71 *****
C600: 72 *****
C600: 73 *****
C600: 74 *****

```

```

C676:C9 96 CMP #196
C677:20 85 RSECT
C678:20 77 PIP
C679:20 77 PIP
C67B:90 C2 BCC
C63F 78 ROPDR
C67D:49 AD BEQ
C67E:10 25 C6A8
C67F:10 25 C6A8
C681:D0 BC BNE
C682:05 43 RSECT
C683:05 43 RSECT
C687:8D 8C LDA
C68A:10 FB C687
C68C:2A BPL
C68D:85 3C STA
C68F:8D 8C LDA
C690:20 77 RSECT
C694:25 3C AND
C696:88 EC DEV
C697:D8 EC BNE
C699:28 93 PIP
C69A:C5 3D C6A8
C69E:A5 48 C63F
C69F:A5 48 C63F
C6A8:C5 41 CMP
C6A2:D8 9B BNE
C6A4:88 9C C642
C6A6:A0 56 LDA
C6A8:8C 8C LDA
C6AD:10 FB C6A8
C6AF:59 D8 C6A2
C6B2:A4 3C LDA
C6B4:88 88 C6A8
C6B6:D8 EF C6A8
C6B8:8C 8C C6B8
C6B9:10 FB C6B8
C6C4:A4 3C LDA
C6C6:91 26 C6B9
C6C8:C8 BNE
C6C9:D8 EF C6B8
C6CB:8C 8C LDA
C6CE:59 D8 C6CB
C6D3:D8 CD C6A2
C6D5:A0 88 LDA
C6D7:A2 56 LDA
C6D9:CA FB C6D7
C6DB:81 26 C6D9
C6DE:5E 88 C6D3
C6E1:2A 127 ROL
C6E2:5E 88 C6D3
C6E3:2A 129 ROL
C6E5:81 26 C6E3
C6E8:C8 BNE
C6E9:D8 EE C6D9
C6EB: C6EB:
C6EB: 134 * Code beyond this point is not
C6EB: 135 * sacred... It may be perverted...
C6EB: 137 * ..any, any, perverted...
C6EB: 137 * ..any, any, perverted...
C6EB:E6 27 INC
C6ED:E6 3D INC
C6EF:A5 3D LDA
C6F4:A5 4F C6E8
C6F6:98 D8 BCC
C6F8:4C 81 JMP
C6FA: 8885 DS

```

;Last byte must be 0

```

12. SWITCHER
C788: 0000
C789: 2 ..... ds $C789-*,0
C790: 3 .....
C791: 4 .....
C792: 5 * Code for switching between banks
C793: 6 * This code appears in both banks of the rom
C794: 7 *
C795: 8 .....
C796: 9 swrt1 sta rombank
C797: 10 swrt1 sta rombank
C798: 11 swrt1 sta rombank
C799: 12 swrt1 sta rombank
C800: 13 swrt1 sta rombank
C801: 14 jmp reset
C802: 15 jmp reset
C803: 16 jmp reset
C804: 17 jmp reset
C805: 18 jmp reset
C806: 19 jmp reset
C807: 20 jmp reset
C808: 21 jmp reset
C809: 22 jmp reset
C810: 23 jmp reset
C811: 24 jmp reset
C812: 25 jmp reset
C813: 26 jmp reset
C814: 27 jmp reset
C815: 28 jmp reset
C816: 29 jmp reset
C817: 30 jmp reset
C818: 31 jmp reset
C819: 32 jmp reset
C820: 33 jmp reset
C821: 34 jmp reset
C822: 35 jmp reset
C823: 36 jmp reset
C824: 37 jmp reset
C825: 38 jmp reset
C826: 39 jmp reset
C827: 40 jmp reset
C828: 41 jmp reset
C829: 42 jmp reset
C830: 43 jmp reset
C831: 44 jmp reset
C832: 45 jmp reset
C833: 46 jmp reset
C834: 47 jmp reset
C835: 48 jmp reset
C836: 49 jmp reset
C837: 50 jmp reset
C838: 51 jmp reset
C839: 52 jmp reset
C840: 53 jmp reset
C841: 54 jmp reset
C842: 55 jmp reset
C843: 56 jmp reset
C844: 57 jmp reset
C845: 58 jmp reset
C846: 59 jmp reset
C847: 60 jmp reset
C848: 61 jmp reset
C849: 62 jmp reset
C850: 63 jmp reset
C851: 64 jmp reset
C852: 65 jmp reset
C853: 66 jmp reset
C854: 67 jmp reset
C855: 68 jmp reset
C856: 69 jmp reset
C857: 70 jmp reset
C858: 71 jmp reset
C859: 72 jmp reset
C860: 73 jmp reset
C861: 74 jmp reset
C862: 75 jmp reset
C863: 76 jmp reset
C864: 77 jmp reset
C865: 78 jmp reset
C866: 79 jmp reset
C867: 80 jmp reset
C868: 81 jmp reset
C869: 82 jmp reset
C870: 83 jmp reset
C871: 84 jmp reset
C872: 85 jmp reset
C873: 86 jmp reset
C874: 87 jmp reset
C875: 88 jmp reset
C876: 89 jmp reset
C877: 90 jmp reset
C878: 91 jmp reset
C879: 92 jmp reset
C880: 93 jmp reset
C881: 94 jmp reset
C882: 95 jmp reset
C883: 96 jmp reset
C884: 97 jmp reset
C885: 98 jmp reset
C886: 99 jmp reset
C887: 100 jmp reset
C888: 101 jmp reset
C889: 102 jmp reset
C890: 103 jmp reset
C891: 104 jmp reset
C892: 105 jmp reset
C893: 106 jmp reset
C894: 107 jmp reset
C895: 108 jmp reset
C896: 109 jmp reset
C897: 110 jmp reset
C898: 111 jmp reset
C899: 112 jmp reset
C900: 113 jmp reset
C901: 114 jmp reset
C902: 115 jmp reset
C903: 116 jmp reset
C904: 117 jmp reset
C905: 118 jmp reset
C906: 119 jmp reset
C907: 120 jmp reset
C908: 121 jmp reset
C909: 122 jmp reset
C910: 123 jmp reset
C911: 124 jmp reset
C912: 125 jmp reset
C913: 126 jmp reset
C914: 127 jmp reset
C915: 128 jmp reset
C916: 129 jmp reset
C917: 130 jmp reset
C918: 131 jmp reset
C919: 132 jmp reset
C920: 133 jmp reset
C921: 134 jmp reset
C922: 135 jmp reset
C923: 136 jmp reset
C924: 137 jmp reset
C925: 138 jmp reset
C926: 139 jmp reset
C927: 140 jmp reset
C928: 141 jmp reset
C929: 142 jmp reset
C930: 143 jmp reset
C931: 144 jmp reset
C932: 145 jmp reset
C933: 146 jmp reset
C934: 147 jmp reset
C935: 148 jmp reset
C936: 149 jmp reset
C937: 150 jmp reset
C938: 151 jmp reset
C939: 152 jmp reset
C940: 153 jmp reset
C941: 154 jmp reset
C942: 155 jmp reset
C943: 156 jmp reset
C944: 157 jmp reset
C945: 158 jmp reset
C946: 159 jmp reset
C947: 160 jmp reset
C948: 161 jmp reset
C949: 162 jmp reset
C950: 163 jmp reset
C951: 164 jmp reset
C952: 165 jmp reset
C953: 166 jmp reset
C954: 167 jmp reset
C955: 168 jmp reset
C956: 169 jmp reset
C957: 170 jmp reset
C958: 171 jmp reset
C959: 172 jmp reset
C960: 173 jmp reset
C961: 174 jmp reset
C962: 175 jmp reset
C963: 176 jmp reset
C964: 177 jmp reset
C965: 178 jmp reset
C966: 179 jmp reset
C967: 180 jmp reset
C968: 181 jmp reset
C969: 182 jmp reset
C970: 183 jmp reset
C971: 184 jmp reset
C972: 185 jmp reset
C973: 186 jmp reset
C974: 187 jmp reset
C975: 188 jmp reset
C976: 189 jmp reset
C977: 190 jmp reset
C978: 191 jmp reset
C979: 192 jmp reset
C980: 193 jmp reset
C981: 194 jmp reset
C982: 195 jmp reset
C983: 196 jmp reset
C984: 197 jmp reset
C985: 198 jmp reset
C986: 199 jmp reset
C987: 200 jmp reset
C988: 201 jmp reset
C989: 202 jmp reset
C990: 203 jmp reset
C991: 204 jmp reset
C992: 205 jmp reset
C993: 206 jmp reset
C994: 207 jmp reset
C995: 208 jmp reset
C996: 209 jmp reset
C997: 210 jmp reset
C998: 211 jmp reset
C999: 212 jmp reset
C1000: 213 jmp reset

```

```

13. IRQBUF
C880: 2 .....
C881: 3 .....
C882: 4 .....
C883: 5 .....
C884: 6 .....
C885: 7 .....
C886: 8 .....
C887: 9 .....
C888: 10 .....
C889: 11 .....
C890: 12 .....
C891: 13 .....
C892: 14 .....
C893: 15 .....
C894: 16 .....
C895: 17 .....
C896: 18 .....
C897: 19 .....
C898: 20 .....
C899: 21 .....
C900: 22 .....
C901: 23 .....
C902: 24 .....
C903: 25 .....
C904: 26 .....
C905: 27 .....
C906: 28 .....
C907: 29 .....
C908: 30 .....
C909: 31 .....
C910: 32 .....
C911: 33 .....
C912: 34 .....
C913: 35 .....
C914: 36 .....
C915: 37 .....
C916: 38 .....
C917: 39 .....
C918: 40 .....
C919: 41 .....
C920: 42 .....
C921: 43 .....
C922: 44 .....
C923: 45 .....
C924: 46 .....
C925: 47 .....
C926: 48 .....
C927: 49 .....
C928: 50 .....
C929: 51 .....
C930: 52 .....
C931: 53 .....
C932: 54 .....
C933: 55 .....
C934: 56 .....
C935: 57 .....
C936: 58 .....
C937: 59 .....
C938: 60 .....
C939: 61 .....
C940: 62 .....
C941: 63 .....
C942: 64 .....
C943: 65 .....
C944: 66 .....
C945: 67 .....
C946: 68 .....
C947: 69 .....
C948: 70 .....
C949: 71 .....
C950: 72 .....
C951: 73 .....
C952: 74 .....
C953: 75 .....
C954: 76 .....
C955: 77 .....
C956: 78 .....
C957: 79 .....
C958: 80 .....
C959: 81 .....
C960: 82 .....
C961: 83 .....
C962: 84 .....
C963: 85 .....
C964: 86 .....
C965: 87 .....
C966: 88 .....
C967: 89 .....
C968: 90 .....
C969: 91 .....
C970: 92 .....
C971: 93 .....
C972: 94 .....
C973: 95 .....
C974: 96 .....
C975: 97 .....
C976: 98 .....
C977: 99 .....
C978: 100 .....
C979: 101 .....
C980: 102 .....
C981: 103 .....
C982: 104 .....
C983: 105 .....
C984: 106 .....
C985: 107 .....
C986: 108 .....
C987: 109 .....
C988: 110 .....
C989: 111 .....
C990: 112 .....
C991: 113 .....
C992: 114 .....
C993: 115 .....
C994: 116 .....
C995: 117 .....
C996: 118 .....
C997: 119 .....
C998: 120 .....
C999: 121 .....
C1000: 122 .....

```

```

Serial & Keyboard buffering
26-JUL-85
PAGE 28
C880: 2 .....
C881: 3 .....
C882: 4 .....
C883: 5 .....
C884: 6 .....
C885: 7 .....
C886: 8 .....
C887: 9 .....
C888: 10 .....
C889: 11 .....
C890: 12 .....
C891: 13 .....
C892: 14 .....
C893: 15 .....
C894: 16 .....
C895: 17 .....
C896: 18 .....
C897: 19 .....
C898: 20 .....
C899: 21 .....
C900: 22 .....
C901: 23 .....
C902: 24 .....
C903: 25 .....
C904: 26 .....
C905: 27 .....
C906: 28 .....
C907: 29 .....
C908: 30 .....
C909: 31 .....
C910: 32 .....
C911: 33 .....
C912: 34 .....
C913: 35 .....
C914: 36 .....
C915: 37 .....
C916: 38 .....
C917: 39 .....
C918: 40 .....
C919: 41 .....
C920: 42 .....
C921: 43 .....
C922: 44 .....
C923: 45 .....
C924: 46 .....
C925: 47 .....
C926: 48 .....
C927: 49 .....
C928: 50 .....
C929: 51 .....
C930: 52 .....
C931: 53 .....
C932: 54 .....
C933: 55 .....
C934: 56 .....
C935: 57 .....
C936: 58 .....
C937: 59 .....
C938: 60 .....
C939: 61 .....
C940: 62 .....
C941: 63 .....
C942: 64 .....
C943: 65 .....
C944: 66 .....
C945: 67 .....
C946: 68 .....
C947: 69 .....
C948: 70 .....
C949: 71 .....
C950: 72 .....
C951: 73 .....
C952: 74 .....
C953: 75 .....
C954: 76 .....
C955: 77 .....
C956: 78 .....
C957: 79 .....
C958: 80 .....
C959: 81 .....
C960: 82 .....
C961: 83 .....
C962: 84 .....
C963: 85 .....
C964: 86 .....
C965: 87 .....
C966: 88 .....
C967: 89 .....
C968: 90 .....
C969: 91 .....
C970: 92 .....
C971: 93 .....
C972: 94 .....
C973: 95 .....
C974: 96 .....
C975: 97 .....
C976: 98 .....
C977: 99 .....
C978: 100 .....
C979: 101 .....
C980: 102 .....
C981: 103 .....
C982: 104 .....
C983: 105 .....
C984: 106 .....
C985: 107 .....
C986: 108 .....
C987: 109 .....
C988: 110 .....
C989: 111 .....
C990: 112 .....
C991: 113 .....
C992: 114 .....
C993: 115 .....
C994: 116 .....
C995: 117 .....
C996: 118 .....
C997: 119 .....
C998: 120 .....
C999: 121 .....
C1000: 122 .....

```


Keyboard buffering

```

C8C0: 162 * The following routine is for reading key-
C8C1: 163 * board from buffers or directly.
C8C2: 164 * Type-ahead buffering only occurs for non auto-
C8C3: 165 * repeat keypresses. When a key is pressed for
C8C4: 166 * auto-repeat the buffer is first emptied, then the
C8C5: 167 * repeated characters are returned.
C8C6: 168 * If the key is used to indicate if a keystroke
C8C7: 169 * is being returned.
C8C8: 170 *
C8C9: 171 *
C8CA: 172 *
C8CB: 173 *
C8CC: 174 *
C8CD: 175 *
C8CE: 176 *
C8CF: 177 *
C8D0: 178 *
C8D1: 179 *
C8D2: 180 *
C8D3: 181 *
C8D4: 182 *
C8D5: 183 *
C8D6: 184 *
C8D7: 185 *
C8D8: 186 *
C8D9: 187 *
C8DA: 188 *
C8DB: 189 *
C8DC: 190 *
C8DD: 191 *
C8DE: 192 *
C8DF: 193 *
C8E0: 194 *
C8E1: 195 *
C8E2: 196 *
C8E3: 197 *
C8E4: 198 *
C8E5: 199 *
C8E6: 200 *
C8E7: 201 *
C8E8: 202 *
C8E9: 203 *
C8EA: 204 *
C8EB: 205 *
C8EC: 206 *
C8ED: 207 *
C8EE: 208 *
C8EF: 209 *
C8F0: 210 *
C8F1: 211 *
C8F2: 212 *
C8F3: 213 *
C8F4: 214 *
C8F5: 215 *
C8F6: 216 *
C8F7: 217 *
C8F8: 218 *
C8F9: 219 *
C8FA: 220 *
C8FB: 221 *
C8FC: 222 *
C8FD: 223 *
C8FE: 224 *
C8FF: 225 *
C900: 226 *
C901: 227 *
C902: 228 *
C903: 229 *
C904: 230 *
C905: 231 *
C906: 232 *
C907: 233 *
C908: 234 *
C909: 235 *
C90A: 236 *
C90B: 237 *
C90C: 238 *
C90D: 239 *
C90E: 240 *
C90F: 241 *
C910: 242 *
C911: 243 *
C912: 244 *
C913: 245 *
C914: 246 *
C915: 247 *
C916: 248 *
C917: 249 *
C918: 250 *
C919: 251 *
C91A: 252 *
C91B: 253 *
C91C: 254 *
C91D: 255 *
C91E: 256 *
C91F: 257 *
C920: 258 *
C921: 259 *
C922: 260 *
C923: 261 *
C924: 262 *
C925: 263 *
C926: 264 *
C927: 265 *
C928: 266 *
C929: 267 *
C92A: 268 *
C92B: 269 *
C92C: 270 *
C92D: 271 *
C92E: 272 *
C92F: 273 *
C930: 274 *
C931: 275 *
C932: 276 *
C933: 277 *
C934: 278 *
C935: 279 *
C936: 280 *
C937: 281 *
C938: 282 *
C939: 283 *
C93A: 284 *
C93B: 285 *
C93C: 286 *
C93D: 287 *
C93E: 288 *
C93F: 289 *
C940: 290 *
C941: 291 *
C942: 292 *
C943: 293 *
C944: 294 *
C945: 295 *
C946: 296 *
C947: 297 *
C948: 298 *
C949: 299 *
C94A: 300 *
C94B: 301 *
C94C: 302 *
C94D: 303 *
C94E: 304 *
C94F: 305 *
C950: 306 *
C951: 307 *
C952: 308 *
C953: 309 *
C954: 310 *
C955: 311 *
C956: 312 *
C957: 313 *
C958: 314 *
C959: 315 *
C95A: 316 *
C95B: 317 *
C95C: 318 *
C95D: 319 *
C95E: 320 *
C95F: 321 *
C960: 322 *
C961: 323 *
C962: 324 *
C963: 325 *
C964: 326 *
C965: 327 *
C966: 328 *
C967: 329 *
C968: 330 *
C969: 331 *
C96A: 332 *
C96B: 333 *
C96C: 334 *
C96D: 335 *
C96E: 336 *
C96F: 337 *
C970: 338 *
C971: 339 *
C972: 340 *
C973: 341 *
C974: 342 *
C975: 343 *
C976: 344 *
C977: 345 *
C978: 346 *
C979: 347 *
C97A: 348 *
C97B: 349 *
C97C: 350 *
C97D: 351 *
C97E: 352 *
C97F: 353 *
C980: 354 *
C981: 355 *
C982: 356 *
C983: 357 *
C984: 358 *
C985: 359 *
C986: 360 *
C987: 361 *
C988: 362 *
C989: 363 *
C98A: 364 *
C98B: 365 *
C98C: 366 *
C98D: 367 *
C98E: 368 *
C98F: 369 *
C990: 370 *
C991: 371 *
C992: 372 *
C993: 373 *
C994: 374 *
C995: 375 *
C996: 376 *
C997: 377 *
C998: 378 *
C999: 379 *
C99A: 380 *
C99B: 381 *
C99C: 382 *
C99D: 383 *
C99E: 384 *
C99F: 385 *
C9A0: 386 *
C9A1: 387 *
C9A2: 388 *
C9A3: 389 *
C9A4: 390 *
C9A5: 391 *
C9A6: 392 *
C9A7: 393 *
C9A8: 394 *
C9A9: 395 *
C9AA: 396 *
C9AB: 397 *
C9AC: 398 *
C9AD: 399 *
C9AE: 400 *
C9AF: 401 *
C9B0: 402 *
C9B1: 403 *
C9B2: 404 *
C9B3: 405 *
C9B4: 406 *
C9B5: 407 *
C9B6: 408 *
C9B7: 409 *
C9B8: 410 *
C9B9: 411 *
C9BA: 412 *
C9BB: 413 *
C9BC: 414 *
C9BD: 415 *
C9BE: 416 *
C9BF: 417 *
C9C0: 418 *
C9C1: 419 *
C9C2: 420 *
C9C3: 421 *
C9C4: 422 *
C9C5: 423 *
C9C6: 424 *
C9C7: 425 *
C9C8: 426 *
C9C9: 427 *
C9CA: 428 *
C9CB: 429 *
C9CC: 430 *
C9CD: 431 *
C9CE: 432 *
C9CF: 433 *
C9D0: 434 *
C9D1: 435 *
C9D2: 436 *
C9D3: 437 *
C9D4: 438 *
C9D5: 439 *
C9D6: 440 *
C9D7: 441 *
C9D8: 442 *
C9D9: 443 *
C9DA: 444 *
C9DB: 445 *
C9DC: 446 *
C9DD: 447 *
C9DE: 448 *
C9DF: 449 *
C9E0: 450 *
C9E1: 451 *
C9E2: 452 *
C9E3: 453 *
C9E4: 454 *
C9E5: 455 *
C9E6: 456 *
C9E7: 457 *
C9E8: 458 *
C9E9: 459 *
C9EA: 460 *
C9EB: 461 *
C9EC: 462 *
C9ED: 463 *
C9EE: 464 *
C9EF: 465 *
C9F0: 466 *
C9F1: 467 *
C9F2: 468 *
C9F3: 469 *
C9F4: 470 *
C9F5: 471 *
C9F6: 472 *
C9F7: 473 *
C9F8: 474 *
C9F9: 475 *
C9FA: 476 *
C9FB: 477 *
C9FC: 478 *
C9FD: 479 *
C9FE: 480 *
C9FF: 481 *

```

65C82 Mini assembler

```

3 *
4 *
5 * Apple //c Mini Assembler
6 *
7 * Got mnemonic, check address mode
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *
32 *
33 *
34 *
35 *
36 *
37 *
38 *
39 *
40 *
41 *
42 *
43 *
44 *
45 *
46 *
47 *
48 *
49 *
50 *
51 *
52 *
53 *
54 *
55 *
56 *
57 *
58 *
59 *
60 *
61 *
62 *
63 *
64 *
65 *
66 *
67 *
68 *
69 *
70 *
71 *
72 *
73 *
74 *
75 *
76 *
77 *
78 *
79 *
80 *
81 *
82 *
83 *
84 *
85 *
86 *
87 *
88 *
89 *
90 *
91 *
92 *
93 *
94 *
95 *
96 *
97 *
98 *
99 *
100 *
101 *
102 *
103 *
104 *
105 *
106 *
107 *
108 *
109 *
110 *
111 *
112 *
113 *
114 *
115 *
116 *
117 *
118 *
119 *
120 *
121 *
122 *
123 *
124 *
125 *
126 *
127 *
128 *
129 *
130 *
131 *
132 *
133 *
134 *
135 *
136 *
137 *
138 *
139 *
140 *
141 *
142 *
143 *
144 *
145 *
146 *
147 *
148 *
149 *
150 *
151 *
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
834 *
835 *
836 *
837 *
838 *
839 *
840 *
841 *
842 *
843 *
844 *
845 *
846 *
847 *
848 *
849 *
850 *
851 *
852 *
853 *
854 *
855 *
856 *
857 *
858 *
859 *
860 *
861 *
862 *
863 *
864 *
865 *
866 *
867 *
868 *
869 *
870 *
871 *
872 *
873 *
874 *
875 *
876 *
877 *
878 *
879 *
880 *
881 *
882 *
883 *
884 *
885 *
886 *
887 *
888 *
889 *
890 *
891 *
892 *
893 *
894 *
895 *
896 *
897 *
898 *
899 *
900 *
901 *
902 *
903 *
904 *
905 *
906 *
907 *
908 *
909 *
910 *
911 *
912 *
913 *
914 *
915 *
916 *
917 *
918 *
919 *
920 *
921 *
922 *
923 *
924 *
925 *
926 *
927 *
928 *
929 *
930 *
931 *
932 *
933 *
934 *
935 *
936 *
937 *
938 *
939 *
940 *
941 *
942 *
943 *
944 *
945 *
946 *
947 *
948 *
949 *
950 *
951 *
952 *
953 *
954 *
955 *
956 *
957 *
958 *
959 *
960 *
961 *
962 *
963 *
964 *
965 *
966 *
967 *
968 *
969 *
970 *
971 *
972 *
973 *
974 *
975 *
976 *
977 *
978 *
979 *
980 *
981 *
982 *
983 *
984 *
985 *
986 *
987 *
988 *
989 *
990 *
991 *
992 *
993 *
994 *
995 *
996 *
997 *
998 *
999 *
1000 *

```

14 MINI	55082 Mini assembler	26-JUL-85	PAGE 33
72	JSR UP	move up 2 lines	
73	JSR UP		
74	DISLIN C983		
75	SHOWINST	Display line & get next instruction	
76	GETINST1	if for prompt	
77	LD A, #A1		
78	STA PROMPT		
79	JSR GETLN2	get a line	
80	BRA DINST	go do the instruction	
81			
82			
83		Compare disassembly of all known opcodes with	
84		if the one typed in until a match is found	
85	GETOP	get opcode	
86	JSR INSDS2	determine mnemonic index	
87	TAX	ix = index	
88	MNEMR,X	get right half of index	
89	CHG A1,OP	does it match entry?	
90	LD A,OP	get left half of index	
91	MNEML,X	get left half of index	
92			
93	bra piskip	skip past pascal stuff	
94	da C98A--",0	Hello I'm the pascal l.0 entry	
95	jmp pwrite	point getting in the way	
96	piskip	just getting in the way	
97			
98	CHP A4H	does it match entry?	
99	BNE NATOP	if no, try next opcode	
100	LD A,C98A	found opcode, check address mode	
101	FORMAT	found opcode, mode format for that	
102	CPY #9D	is it relative?	
103	BEQ REL	if yes, calc relative address	
104	CHP FORMAT	does mode match?	
105	CHP FORMINST	if yes, move instruction to memory	
106	DEC A1H	try next opcode	
107	BNE GETOP	if go try it	
108	INC ASL	else try next format	
109	DEC YSAV1		
110	BEQ GETOP	if go try next format	
111			
112		Point to the error with a caret, beep, and fall	
113		into the mini-assembler.	
114			
115	MINIERR LDY YSAV	get position	
116	ERR2 TYA		
117	TAX		
118	ERR3		
119	PBR2		
120	LD A,#DE	to point to error	
121	JSR CODE		
122	JSR BELL	beep cause we're mad	
123	BRA GETINST1	try again	
124			
125		Read a line of input. If prefaced with " ", decode	
126		mnemonic. If "a" do monitor command. Otherwise parse	
127		hex address before decoding mnemonic.	
128			
129	DOINST	clear mode	
130	JSR ZMODE		
131	LD A,#80	if blank	
132	BEQ DOLIN	if blank char in line	
133	CHP #8D	if go attempt disassembly	
134	BNE GETI1	is it return?	
135		if no, continue to Monitor	
136		else return to Monitor	
137	GETI1	parse hexadecimal input	
138	CHP #93	look for "ADDR:"	
139	DOERR2	if no ":", display error	
140	TAX	if nonzero if address entered	
141	ERR2	if no "ADDR:", display error	
142			

[illegible]


```

15 SCROLLING
CBA8:D0 F9 CBB9
CBA2:70 04 CBA8
CBA6:01 20 CBA8
CBA8:AD 54 C8
CBAE:AC F8 05 CBB4
CBAE:10 04 CBB4
CBA8:10 20 CBB4
CBA8:10 2A CBB4
CBA4:00 2A CBB4
CBB5:10 F9 CBB8
CBB7:00 B4 CBB4
CBB9:20 A0 FC
CBBF:20 22 FC
CBB8:FA
CBB0:FA
CBB0:00

74 BNE SCRLVEN
75 SKPLFT Y
76 STA (BASL),Y
77 LDA TXTPAGE1
78 SKPLFT
79 LDV TEMPY
80 BCS SKRPT
81 SCLRODD
82 DEV
83 SKRPT
84 BRA SCRLDD
85
86 *
87 SCRL3
88
89
90 PLX
91 SEV1
RTS

;do all but last even byte
;odd left edge, skip this byte
;restore width
;even right edge, skip this byte
;scroll next line
;clear current line
;pull status off stack
;restore X
;done!!!

BNE SCRLVEN
BVS SKPLFT Y
STA (BASL),Y
LDA TXTPAGE1
LDV TEMPY
BCS SKRPT
SCLRODD
DEV
SKRPT
BRA SCRLDD
*
SCRL3
PLX
SEV1
RTS

26-JUL-85 PAGE 39
Apple //c Video firmware
15 SCROLLING
CBA2:70 04 CBA8
CBA6:01 20 CBA8
CBA8:AD 54 C8
CBAE:AC F8 05 CBB4
CBAE:10 04 CBB4
CBA8:10 20 CBB4
CBA8:10 2A CBB4
CBA4:00 2A CBB4
CBB5:10 F9 CBB8
CBB7:00 B4 CBB4
CBB9:20 A0 FC
CBBF:20 22 FC
CBB8:FA
CBB0:FA
CBB0:00

93 * DOCLR is called by CLRDL. It decides whether
94 * to do a (quiet) 40 or 80 column clear to end of line.
95 *
96 *
97 DOCLR BIT RD8VID
98 STA (BASL),Y
99 CLR40
100 CLR40
101 CLR40
102 CLR40
103
104 *
105 CLRHALF
106 LDA
107 LDA
108 LDA INVFLG
109 AND #0A0
110 BRA CLR2
111
112 *
113 CLR00
114 PHX
115 PHA
116 PHA
117 SEC
118 SBC
119 TYA
120 LSR A
121 TAY
122 PLA
123 ROR A
124 BCS CLR0
125 BPL CLR0
126
127 INY
128 CLR0
129 CBFC
130 CBFC
131 CLR2
132
133 INX
134 CLR1
135 TNA
136 INX
137
138 BNE PLX
139 CLR3
140
141 *
142 CLRPORT
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

15 SCROLLING
CDB5:A0 00
CDB9:8C 7B 84
CDBC:AC 7B 85
CDBF:60
CC00:
43

Apple //c Video firmware
288 GETCUR3
289 LDY #0
290 STY OLDCH
291 LDY DURCH
292 GETCURX RTS
43 INCLUDE ESCAPE

```

PAGE 43

26-JUL-85

```

:yes, peg CH to 0
:iget cursor
:and fly...

```

16 ESCAPE

```

CC00:
CC01:
CC02:
CC03:
CC04:
CC05:
CC06:
CC07:
CC08:
CC09:
CC0A:
CC0B:
CC0C:
CC0D:
CC0E:
CC0F:
CC10:
CC11:
CC12:
CC13:
CC14:
CC15:
CC16:
CC17:
CC18:
CC19:
CC1A:
CC1B:
CC1C:
CC1D:
CC1E:
CC1F:
CC20:
CC21:
CC22:
CC23:
CC24:
CC25:
CC26:
CC27:
CC28:
CC29:
CC2A:
CC2B:
CC2C:
CC2D:
CC2E:
CC2F:
CC30:
CC31:
CC32:
CC33:
CC34:
CC35:
CC36:
CC37:
CC38:
CC39:
CC3A:
CC3B:
CC3C:
CC3D:
CC3E:
CC3F:
CC40:
CC41:
CC42:
CC43:
CC44:
CC45:
CC46:
CC47:
CC48:
CC49:
CC4A:
CC4B:
CC4C:
CC4D:
CC4E:
CC4F:
CC50:
CC51:
CC52:
CC53:
CC54:
CC55:
CC56:
CC57:
CC58:
CC59:
CC5A:
CC5B:
CC5C:
CC5D:
CC5E:
CC5F:
CC60:
CC61:
CC62:
CC63:
CC64:
CC65:
CC66:
CC67:
CC68:
CC69:
CC6A:
CC6B:
CC6C:
CC6D:
CC6E:
CC6F:
CC70:
CC71:
CC72:
CC73:
CC74:
CC75:
CC76:
CC77:
CC78:
CC79:
CC7A:
CC7B:
CC7C:
CC7D:
CC7E:
CC7F:
CC80:
CC81:
CC82:
CC83:
CC84:
CC85:
CC86:
CC87:
CC88:
CC89:
CC8A:
CC8B:
CC8C:
CC8D:
CC8E:
CC8F:
CC90:
CC91:
CC92:
CC93:
CC94:
CC95:
CC96:
CC97:
CC98:
CC99:
CC9A:
CC9B:
CC9C:
CC9D:
CC9E:
CC9F:
CCA0:
CCA1:
CCA2:
CCA3:
CCA4:
CCA5:
CCA6:
CCA7:
CCA8:
CCA9:
CCAA:
CCAB:
CCAC:
CCAD:
CCAE:
CCAF:
CCB0:
CCB1:
CCB2:
CCB3:
CCB4:
CCB5:
CCB6:
CCB7:
CCB8:
CCB9:
CCBA:
CCBB:
CCBC:
CCBD:
CCBE:
CCBF:
CCC0:
CCC1:
CCC2:
CCC3:
CCC4:
CCC5:
CCC6:
CCC7:
CCC8:
CCC9:
CCC0:
CCC1:
CCC2:
CCC3:
CCC4:
CCC5:
CCC6:
CCC7:
CCC8:
CCC9:

```

Apple //c Video firmware

PAGE 44

26-JUL-85

```

2 * START AN ESCAPE SEQUENCE:
3 * WE HANDLE THE FOLLOWING ONES:
4 * A - HOME - CLEAR
5 * B - Cursor right
6 * C - Cursor left
7 * D - Cursor down
8 * E - Cursor up
9 * F - CLR TO EOS
10 *
11 * 1 - Up Arrow - CURSOR UP (stay escape)
12 * J, Lt Arrow - CURSOR LEFT (stay escape)
13 * K, Rt Arrow - CURSOR RIGHT (stay escape)
14 * M - CURSOR DOWN (stay escape)
15 * 4 - GOTD 40 COLUMN MODE
16 * 8 - GOTD 80 COLUMN MODE
17 * CTL-D- Disable the printing of control chars
18 * CTL-E- Enable the printing of control chars
19 *
20 * CTL-G- QUIT (PR#0/IN#0)
21 *
22 * LDA ESCCHAR,Y
23 * save index
24 * JSR CTLCHAR
25 * restore index
26 * PLY #WHI
27 * BCS ESCRDKEY
28 *
29 * This is the entry point called by RDKKEY iff escapes
30 * are enabled and an escape is encountered. The next
31 * keypress is read and processed. If it is a key that
32 * terminates escape mode, a new key is read by ESCRDKEY.
33 * If escape mode should not be terminated, MEMESC is
34 * called again.
35 *
36 * MEMESC
37 * PHA
38 * PHA #400
39 * PHA #0B
40 * JSR STORY
41 * JSR XBTKBD
42 * BPL ESC0
43 * PLA
44 * CLRABD
45 * JSR UPSHIFT
46 * LDY #ESCNUM
47 * CMP ESCTAB,Y
48 * BEQ ESC3
49 * DEY
50 * BPL ESC2
51 *
52 * End of escape sequence, read next character.
53 * This is initially called by RDCCHAR which is usually called
54 * by GETLN to read characters with escapes enabled.
55 *
56 * ESCRDKEY LDA #M_CTL
57 * VNOIE
58 * JSR RDKKEY
59 * JMP NOESCAPE
60 *
61 * When in escape mode, the characters in ESCTAB (high
62 * bits set), are mapped into the characters in ESCCHAR.
63 * These characters are then executed by a call to CTLCHAR.
64 *
65 * CTLCHAR looks up a character in the table starting at
66 * table of routine addresses, CTLADR. If the character is
67 * not in the table, call to VNOIE.
68 * Character is BS, LF, CR, or BEL.
69 *
70 * NOTE: CTLON and CTLOFF are not accessible except through
71 * and escape sequence
72 *

```


17 PASCAL	Video firmware Pascal stuff	26-JUL-85	PAGE 51
CEB1:AA	3 PSTATUS	TAX	!is request code = 0?
CEB2:FA	4 BNE	PIORDY	!>yes, >for output
CEB4:CA	5 DEX		!>check for any input
CEB5:D0	6 BNE	PTERR	!>bad request, return error
CEB7:20	7 JSR	XBITKBD	!test keyboard
CEB8:30	8 BNE	PHOTRDY	!no keyboard
CEB9:80	9 PIORDY		!good return
CEBD:60	10 PHOTRDY	RTS	!good return
CEBE:A2	11 PSTERR	LDX	!else flag error
CEC0:10	12 PHOTRDY	CLC	
CEC1:60	13 PHOTRDY	RTS	
CEC2:	14 * PASCAL OUTPUT:		
CEC2:	15 * PASCAL OUTPUT:		
CEC2:	16 * PASCAL OUTPUT:		
CEC2:	17 PHRITE	RTS	
CEC2:89	18 DRA	#000	!turn on high bit
CEC4:AA	19 TAX		!save character
CEC5:20	20 JSR	PSETUP2	!SETUP 2P STUFF, don't set ROM
CEC5:20	21 JSR	PMOUSE	!ARE WE DOING OUTDAY?
CEC6:20	22 BIT	VMODE	!no
CEC6:20	23 BNE	GETX	!>Doing X or Y?
CEC6:8A	24 TAX		!now check for control char
CEC6:80	25 BIT		!is it control?
CEC6:80	26 BIT		!yes, so control
CEC6:80	27 LDX	DURCH	!check for control
CEC6:80	28 BIT	INVLG	!check for inverse
CEC6:80	29 BNE	PMR1	!normal, go store it
CEC6:80	30 BNE	#0FF	!normal, go store it
CEC6:80	31 JSR	STORE	!normal, go store it
CEC6:80	32 JSR	STORE	!normal, go store it
CEC6:80	33 JSR	STORE	!normal, go store it
CEC6:80	34 CPY	WIDWIDTH	!normal, go store it
CEC6:80	35 BCC	PMR2	!normal, go store it
CEC6:80	36 JSR	SETROM	!normal, go store it
CEC6:80	37 JSR	SETROM	!normal, go store it
CEC6:80	38 JSR	SETROM	!normal, go store it
CEC6:80	39 PHRITERET	JSR	RESETLC
CEC6:80	40 PHRITERET	JSR	PASINVERT
CEC6:80	41 PRET	LDX	#00
CEC6:80	42 PRET	RTS	
CEC6:80	43 * HANDLE GOTDXY STUFF:		
CEC6:80	44 * HANDLE GOTDXY STUFF:		
CEC6:80	45 * HANDLE GOTDXY STUFF:		
CEC6:80	46 GETX	RTS	
CEC6:80	47 JSR	PASINVERT	
CEC6:80	48 TAX		!turn off cursor
CEC6:80	49 JSR	SETROM	!get character
CEC6:80	50 SBC	#160	!MAKE BINARY
CEC6:80	51 BIT	XCOORD	!doing X?
CEC6:80	52 BNE	PSETX	!>yes, set it
CEC6:80	53 * Set Y and do the GOTDXY		
CEC6:80	54 * Set Y and do the GOTDXY		
CEC6:80	55 * Set Y and do the GOTDXY		
CEC6:80	56 * Set Y and do the GOTDXY		
CEC6:80	57 GETY	RTS	
CEC6:80	58 STA	PASCALC	!calc base addr
CEC6:80	59 LDX	XCOORD	!set proper cursors
CEC6:80	60 JSR	GETCUR2	!turn off gotdxy
CEC6:80	61 JSR	GETCUR2	!turn off gotdxy
CEC6:80	62 JSR	GETCUR2	!turn off gotdxy
CEC6:80	63 JSR	GETCUR2	!turn off gotdxy
CEC6:80	64 JSR	GETCUR2	!turn off gotdxy
CEC6:80	65 PCTL		!turn off cursor
CEC6:80	66 JSR	SETROM	!get character
CEC6:80	67 JSR	SETROM	!get character
CEC6:80	68 JSR	SETROM	!get character
CEC6:80	69 JSR	SETROM	!get character
CEC6:80	70 JSR	SETROM	!get character
CEC6:80	71 JSR	SETROM	!get character
CEC6:80	72 JSR	SETROM	!get character
CEC6:80	73 * START THE GOTDXY SEQUENCE:		

17 PASCAL	Video firmware Pascal stuff	26-JUL-85	PAGE 52
CFB2:	74 * STARTIX	RTS	
CFB2:	75 * STARTIX	RTS	
CFB2:	76 * STARTIX	RTS	
CFB2:80	77 JSR	PMOUSE	!Set mode to pascal
CFB2:80	78 JSR	PMOUSE	!without mouse characters
CFB2:80	79 JSR	PMOUSE	!without mouse characters
CFB2:80	80 JSR	PMOUSE	!without mouse characters
CFB2:80	81 JSR	PMOUSE	!without mouse characters
CFB2:80	82 JSR	PMOUSE	!without mouse characters
CFB2:80	83 JSR	PMOUSE	!without mouse characters
CFB2:80	84 JSR	PMOUSE	!without mouse characters
CFB2:80	85 JSR	PMOUSE	!without mouse characters
CFB2:80	86 JSR	PMOUSE	!without mouse characters
CFB2:80	87 JSR	PMOUSE	!without mouse characters
CFB2:80	88 JSR	PMOUSE	!without mouse characters
CFB2:80	89 JSR	PMOUSE	!without mouse characters
CFB2:80	90 JSR	PMOUSE	!without mouse characters
CFB2:80	91 JSR	PMOUSE	!without mouse characters
CFB2:80	92 JSR	PMOUSE	!without mouse characters
CFB2:80	93 JSR	PMOUSE	!without mouse characters
CFB2:80	94 JSR	PMOUSE	!without mouse characters
CFB2:80	95 JSR	PMOUSE	!without mouse characters
CFB2:80	96 JSR	PMOUSE	!without mouse characters
CFB2:80	97 JSR	PMOUSE	!without mouse characters
CFB2:80	98 JSR	PMOUSE	!without mouse characters
CFB2:80	99 JSR	PMOUSE	!without mouse characters
CFB2:80	100 JSR	PMOUSE	!without mouse characters
CFB2:80	101 JSR	PMOUSE	!without mouse characters
CFB2:80	102 JSR	PMOUSE	!without mouse characters
CFB2:80	103 JSR	PMOUSE	!without mouse characters
CFB2:80	104 JSR	PMOUSE	!without mouse characters
CFB2:80	105 JSR	PMOUSE	!without mouse characters
CFB2:80	106 JSR	PMOUSE	!without mouse characters
CFB2:80	107 JSR	PMOUSE	!without mouse characters
CFB2:80	108 JSR	PMOUSE	!without mouse characters
CFB2:80	109 JSR	PMOUSE	!without mouse characters
CFB2:80	110 JSR	PMOUSE	!without mouse characters
CFB2:80	111 JSR	PMOUSE	!without mouse characters
CFB2:80	112 JSR	PMOUSE	!without mouse characters
CFB2:80	113 JSR	PMOUSE	!without mouse characters
CFB2:80	114 JSR	PMOUSE	!without mouse characters
CFB2:80	115 JSR	PMOUSE	!without mouse characters
CFB2:80	116 JSR	PMOUSE	!without mouse characters
CFB2:80	117 JSR	PMOUSE	!without mouse characters
CFB2:80	118 JSR	PMOUSE	!without mouse characters
CFB2:80	119 JSR	PMOUSE	!without mouse characters
CFB2:80	120 JSR	PMOUSE	!without mouse characters
CFB2:80	121 JSR	PMOUSE	!without mouse characters
CFB2:80	122 JSR	PMOUSE	!without mouse characters
CFB2:80	123 JSR	PMOUSE	!without mouse characters
CFB2:80	124 JSR	PMOUSE	!without mouse characters
CFB2:80	125 JSR	PMOUSE	!without mouse characters
CFB2:80	126 JSR	PMOUSE	!without mouse characters
CFB2:80	127 JSR	PMOUSE	!without mouse characters
CFB2:80	128 JSR	PMOUSE	!without mouse characters
CFB2:80	129 JSR	PMOUSE	!without mouse characters
CFB2:80	130 JSR	PMOUSE	!without mouse characters
CFB2:80	131 JSR	PMOUSE	!without mouse characters
CFB2:80	132 JSR	PMOUSE	!without mouse characters
CFB2:80	133 JSR	PMOUSE	!without mouse characters
CFB2:80	134 JSR	PMOUSE	!without mouse characters
CFB2:80	135 JSR	PMOUSE	!without mouse characters
CFB2:80	136 JSR	PMOUSE	!without mouse characters
CFB2:80	137 JSR	PMOUSE	!without mouse characters
CFB2:80	138 JSR	PMOUSE	!without mouse characters
CFB2:80	139 JSR	PMOUSE	!without mouse characters
CFB2:80	140 JSR	PMOUSE	!without mouse characters
CFB2:80	141 JSR	PMOUSE	!without mouse characters
CFB2:80	142 JSR	PMOUSE	!without mouse characters
CFB2:80	143 JSR	PMOUSE	!without mouse characters
CFB2:80	144 JSR	PMOUSE	!without mouse characters
CFB2:80	145 JSR	PMOUSE	!without mouse characters

[illegible]

19 AUTOST1	Apple //c F8 monitor firmware	26-JUL-85	PAGE 55
F801:4A	3 PLOT	LSR A	:Y-COORD/2
F801:80	4 PIP	LSR A	:SAVE LSB IN CARRY
F802:20 47 FB	5	JSR GRASCALC	:CALC BASE ADDR IN GBASL,H
F805:20 47 FB	6	PLP	:RESTORE LSB FROM CARRY
F806:90 47 FB	7	PLP	:MASK #BF IF EVEN
F808:90 47 FB	8	BCC #B8	:MASK #BF IF ODD
F80A:60 47 FB	9	ADC #B8	
F80C:85 2E	10 RTMASK	STA MASK	
F80E:B1 56	11 PLOT1	LDA (GBASL),Y	
F810:50 56	12	LDX COLOR	:XOR MASK
F812:50 56	13	LDX MASK	:XOR DATA
F814:50 56	14	EDR (GBASL),Y	:XOR DATA
F816:31 26	15	STA (GBASL),Y	:TO DATA
F818:60	16	RTS	
F819:20 80 FB	17	JSR PLOT	:PLOT SQUARE
F81C:CA 2C	18	CPY H2	:DONE?
F81E:B0 11	19	BCC RTS1	:YES, RETURN
F820:C0	20	INY	:NO, INCR INDEX (X-COORD)
F821:20 80 FB	21	JSR PLOT1	:PLOT NEXT SQUARE
F823:20 80 FB	22	ADC #B8	:NEW Y-COORD
F825:20 80 FB	23	BCC RTS1	:NEW Y-COORD
F827:20 80 FB	24	PLP	:SAVE ON STACK
F829:20 80 FB	25	JSR PLOT	:PLOT SQUARE
F82C:60	26	CLP	
F82D:C5 2D	27	PLA V2	:DONE?
F82F:30 5F	28	BCC VLINEZ	:NO, LOOP.
F831:30 5F	29	BCC RTS1	
F832:A0 2F	30	LDY #2F	:MAX Y, FULL SCRIN CLR
F834:D0 92	31	BNE CLRSC2	:ALWAYS TAKEN
F836:A0 27	32 CLRSC2	LDY #27	:MAX Y, TOP SCRIN CLR
F838:90 27	33	STY V2	:FOR BOTTOM CORD
F83A:80 27	34 CLRSC2	LDY #27	:FOR LINE CORD
F83C:A0 27	35	LDY #27	:RIGHTMOST X-COORD (COLUMN)
F83E:85 38	36 CLRSC3	LDY #38	:TOP CORD FOR VLINE CALLS
F840:20 28 FB	37	STA COLOR	:CLEAR COLOR (BLACK)
F842:90 56	38	JSR VLINE	:DRAW VLINE
F844:90 56	39	BPL CLRSC3	:NEXT X-COORD
F846:60	40	RTS	:LOOP UNTIL DONE.
F847:	41		
F847:40	42	PHA	:FOR INPUT
F848:3A 93	43	LSR A	:DEFH
F84B:80 84	44	ORA #84	
F84D:85 27	45	ORA GBASH	:GENERATE GBASH=000001FG
F84F:60 19	46	STA	:AND GBASL-HDEDE000
F850:20 19	47	AND #19	
F851:60 9F	48	ADC #9F	
F853:85 26	49	STB GBASL	
F855:85 26	50	ASL A	
F857:85 26	51	ASL A	
F859:8A	52	ASL A	
F85B:85 26	53	ASL A	
F85D:85 26	54	ASL A	
F85F:80	55	RTS	
F861:18	56	CLC	:INCREMENT COLOR BY 3
F862:60 83	57	AND #83	
F864:20 8F	58	AND #8F	:SETS COLOR=17FA MOD 16
F866:90 38	59	ASL A	
F868:90 38	60	ASL A	:BOTH HALF BYTES OF COLOR EQUAL
F86A:8A	61	ASL A	
F86B:8A	62	ASL A	
F86D:85 38	63	ASL A	
F86F:85 38	64	ASL A	
F871:	65	RTS	
F871:	66		
F871:	67		
F871:	68		
F871:	69		
F871:	70		
F871:	71		
F871:	72		
F871:	73		

19 AUTOST1	Apple //c F8 monitor firmware	26-JUL-85	PAGE 56
F871:4A	74 SCRIN	LSR A	:READ SCREEN Y-COORD/2
F871:80	75 PIP	PIP	:SAVE LSB (CARRY)
F873:20 47 FB	76	JSR GRASCALC	:CALC BASE ADDRESS
F875:20 47 FB	77	LDA (GBASL),Y	:GET BYTE LSB FROM CARRY
F877:50 84	78	BCC RTMSKZ	:IF EVEN, USE LD H
F879:50 84	79	SCRIN2	
F87B:4A	80	LSR A	
F87C:4A	81	LSR A	:SHIFT HIGH HALF BYTE DOWN
F87D:4A	82	LSR A	
F87F:4A	83	LSR A	:MASK 4-BITS
F881:60	84	RTMSKZ	
F883:	85	RTS	
F885:	86		
F887:4A	87	LDX PCL	:PRINT PCL,H
F889:4A	88	LDY PCH	
F88B:4A	89	LDX PCL	
F88D:4A	90	LDY PCH	:FOLLOWED BY A BLANK
F88F:4A	91	LDX PCL	:GET OP CODE
F891:4A	92	LDX PCL	:EVEN/ODD TEST
F893:4A	93	LDX PCL	:BIT 1 TEST
F895:4A	94	LDX PCL	:XXXXXXXX11 INVALID OP
F897:4A	95	LDX PCL	:MASK BITS
F899:4A	96	LDX PCL	:LSB INTO CARRY FOR L/R TEST
F89B:4A	97	LDX PCL	
F89D:4A	98	LDX PCL	:SET FORMAT INDEX BYTE
F89F:4A	99	LDX PCL	:L/R BYTE ON CARRY
F8A1:4A	100	LDX PCL	:SUBSTITUTE #FC FOR INVALID OPS
F8A3:4A	101	LDX PCL	:SET PRINT FORMAT INDEX TO 0
F8A5:4A	102	LDX PCL	
F8A7:4A	103	LDX PCL	:INDEX INTO PRINT FORMAT TABLE
F8A9:4A	104	LDX PCL	:SAVE FOR ADDRESS FIELD FORMATTING
F8AB:4A	105	LDX PCL	:MASK FOR 2-BIT LENGTH
F8AD:4A	106	LDX PCL	
F8AF:4A	107	LDX PCL	
F8B1:4A	108	LDX PCL	:get index for new opcodes
F8B3:4A	109	LDX PCL	:found new op (or no op)
F8B5:4A	110	LDX PCL	:SAVE IT
F8B7:4A	111	LDX PCL	:OPCODE TO A AGAIN
F8B9:4A	112	LDX PCL	
F8BB:4A	113	LDX PCL	
F8BD:4A	114	LDX PCL	
F8BF:4A	115	LDX PCL	
F8C1:4A	116	LDX PCL	
F8C3:4A	117	LDX PCL	
F8C5:4A	118	LDX PCL	
F8C7:4A	119	LDX PCL	
F8C9:4A	120	LDX PCL	
F8CB:4A	121	LDX PCL	
F8CD:4A	122	LDX PCL	
F8CF:4A	123	LDX PCL	
F8D1:4A	124	LDX PCL	
F8D3:4A	125	LDX PCL	
F8D5:4A	126	LDX PCL	
F8D7:4A	127	LDX PCL	
F8D9:4A	128	LDX PCL	
F8DB:4A	129	LDX PCL	
F8DD:4A	130	LDX PCL	
F8DF:4A	131	LDX PCL	
F8E1:4A	132	LDX PCL	
F8E3:4A	133	LDX PCL	
F8E5:4A	134	LDX PCL	
F8E7:4A	135	LDX PCL	
F8E9:4A	136	LDX PCL	
F8EB:4A	137	LDX PCL	
F8ED:4A	138	LDX PCL	
F8EF:4A	139	LDX PCL	
F8F1:4A	140	LDX PCL	
F8F3:4A	141	LDX PCL	
F8F5:4A	142	LDX PCL	
F8F7:4A	143	LDX PCL	
F8F9:4A	144	LDX PCL	

F871:4A	74 SCRIN	LSR A	:READ SCREEN Y-COORD/2
F871:80	75 PIP	PIP	:SAVE LSB (CARRY)
F873:20 47 FB	76	JSR GRASCALC	:CALC BASE ADDRESS
F875:20 47 FB	77	LDA (GBASL),Y	:GET BYTE LSB FROM CARRY
F877:50 84	78	BCC RTMSKZ	:IF EVEN, USE LD H
F879:50 84	79	SCRIN2	
F87B:4A	80	LSR A	
F87C:4A	81	LSR A	:SHIFT HIGH HALF BYTE DOWN
F87D:4A	82	LSR A	
F87F:4A	83	LSR A	:MASK 4-BITS
F881:60	84	RTMSKZ	
F883:	85	RTS	
F885:	86		
F887:4A	87	LDX PCL	:PRINT PCL,H
F889:4A	88	LDY PCH	
F88B:4A	89	LDX PCL	
F88D:4A	90	LDY PCH	:FOLLOWED BY A BLANK
F88F:4A	91	LDX PCL	:GET OP CODE
F891:4A	92	LDX PCL	:EVEN/ODD TEST
F893:4A	93	LDX PCL	:BIT 1 TEST
F895:4A	94	LDX PCL	:XXXXXXXX11 INVALID OP
F897:4A	95	LDX PCL	:MASK BITS
F899:4A	96	LDX PCL	:LSB INTO CARRY FOR L/R TEST
F89B:4A	97	LDX PCL	
F89D:4A	98	LDX PCL	:SET FORMAT INDEX BYTE
F89F:4A	99	LDX PCL	:L/R BYTE ON CARRY
F8A1:4A	100	LDX PCL	:SUBSTITUTE #FC FOR INVALID OPS
F8A3:4A	101	LDX PCL	:SET PRINT FORMAT INDEX TO 0
F8A5:4A	102	LDX PCL	
F8A7:4A	103	LDX PCL	:INDEX INTO PRINT FORMAT TABLE
F8A9:4A	104	LDX PCL	:SAVE FOR ADDRESS FIELD FORMATTING
F8AB:4A	105	LDX PCL	:MASK FOR 2-BIT LENGTH
F8AD:4A	106	LDX PCL	
F8AF:4A	107	LDX PCL	
F8B1:4A	108	LDX PCL	:get index for new opcodes
F8B3:4A	109	LDX PCL	:found new op (or no op)
F8B5:4A	110	LDX PCL	:SAVE IT
F8B7:4A	111	LDX PCL	:OPCODE TO A AGAIN
F8B9:4A	112	LDX PCL	
F8BB:4A	113	LDX PCL	
F8BD:4A	114	LDX PCL	
F8BF:4A	115	LDX PCL	
F8C1:4A	116	LDX PCL	
F8C3:4A	117	LDX PCL	
F8C5:4A	118	LDX PCL	
F8C7:4A	119	LDX PCL	
F8C9:4A	120	LDX PCL	
F8CB:4A	121	LDX PCL	
F8CD:4A	122	LDX PCL	
F8CF:4A	123	LDX PCL	
F8D1:4A	124	LDX PCL	
F8D3:4A	125	LDX PCL	
F8D5:4A	126	LDX PCL	
F8D7:4A	127	LDX PCL	
F8D9:4A	128	LDX PCL	
F8DB:4A	129	LDX PCL	
F8DD:4A	130	LDX PCL	
F8DF:4A	131	LDX PCL	
F8E1:4A	132	LDX PCL	
F8E3:4A	133	LDX PCL	
F8E5:4A	134	LDX PCL	
F8E7:4A	135	LDX PCL	
F8E9:4A	136	LDX PCL	
F8EB:4A	137	LDX PCL	
F8ED:4A	138	LDX PCL	
F8EF:4A	139	LDX PCL	
F8F1:4A	140	LDX PCL	
F8F3:4A	141	LDX PCL	
F8F5:4A	142	LDX PCL	
F8F7:4A	143	LDX PCL	
F8F9:4A	144	LDX PCL	

Apple //c F8 monitor firmware
(x=INDEX)

19 AUTOST1

26-JUL-85

PAGE 57

Apple //c F8 monitor firmware

19 AUTOST1

```

F9E9:68 145 PLA
F9EA:A8 146 TAY
F9EB:89 C8 F9 LDA MNEML,Y
F9EC:85 2C 148 STA LNMEM
F9ED:89 28 FA LDA MNEML,Y
F9EE:89 88 150 SCL MNEML
F9EF:89 88 151 PRMN1 LDA #000
F9F0:89 88 152 LDY #485
F9F1:A8 05 153 PRMN2 ASL MNEML
F9F2:85 2D 154 ROL LNMEM
F9F3:85 2C 155 ROL LNMEM
F9F4:8A 156 LCL A
F9F5:8A 157 BNE PRMN2
F9F6:8A 158 BNE PRMN2
F9F7:8A 159 ADC #00F
F9F8:8A 160 JSR COUT
F9F9:8A 161 DEX
F9FA:8A 162 BNE PRMN1
F9FB:8A 163 JSR PRBLNK
F9FC:8A 164 JSR PRBLNK
F9FD:8A 165 LDX #00F
F9FE:8A 166 CPX #483
F9FF:8A 167 PRADR1
F9A0:8A 168 BEQ PRADR2
F9A1:8A 169 ASL FORMAT
F9A2:8A 170 BCC PRADR3
F9A3:8A 171 LCL A
F9A4:8A 172 LDA CHAR2-1,X
F9A5:8A 173 BEQ PRADR3
F9A6:8A 174 JSR COUT
F9A7:8A 175 BNE PRADR1
F9A8:8A 176 RTS
F9A9:8A 177 *
F9AA:8A 178 PRADR4
F9AB:8A 179 DEY
F9AC:8A 180 BNE PRADR2
F9AD:8A 181 LDA MNEML
F9AE:8A 182 LDA FORMAT
F9AF:8A 183 CMP #4E8
F9B0:8A 184 LDA (PCL),Y
F9B1:8A 185 BCC PRADR4
F9B2:8A 186 JSR PCADJ3
F9B3:8A 187 INX
F9B4:8A 188 BNE PRNTYX
F9B5:8A 189 INY
F9B6:8A 190 PRNTYX
F9B7:8A 191 PRNTYX
F9B8:8A 192 PRNTYX
F9B9:8A 193 JMP
F9BA:8A 194 *
F9BB:8A 195 LDX
F9BC:8A 196 PRBL2
F9BD:8A 197 PRBL3
F9BE:8A 198 JSR COUT
F9BF:8A 199 BNE PRBL2
F9C0:8A 200 RTS
F9C1:8A 201 *
F9C2:8A 202 PCADJ
F9C3:8A 203 PCADJ2
F9C4:8A 204 PCADJ3
F9C5:8A 205 TAX
F9C6:8A 206 BPL PCADJ4
F9C7:8A 207 DEY
F9C8:8A 208 PCADJ4
F9C9:8A 209 PCADJ4
F9CA:8A 210 INY
F9CB:8A 211 RTS
F9CC:8A 212 *
F9CD:8A 213 : FMT1 BYTES:
F9CE:8A 214 THEN RIGHT HALF BYTE
F9CF:8A 215 : IF Y=1

```

19 AUTOSTT1	Apple //c FB monitor firmware	26-JUL-85	PAGE 59	19 AUTOSTT1	Apple //c FB monitor firmware	26-JUL-85	PAGE 60
F9A6:80	287 FMT2	DFB \$88		F9EB:AD	358 DFB \$AD		
F9A7:21	288 DFB \$29	:ERR		F9EC:29	359 DFB \$29		
F9A8:81	289 DFB \$81	:IMM		F9ED:8A	360 DFB \$8A		
F9A9:22	290 DFB \$22	:Z-PAGE		F9EE:1C	361 DFB \$1C		
F9AA:52	291 DFB \$52	:ZPAG,X		F9EF:6B	362 DFB \$6B	: (C) FORMAT	
F9AB:4D	292 DFB \$4D	:ZPAG,X		F9F0:15	363 DFB \$15		
F9AC:91	293 DFB \$91	:ZPAG,X		F9F1:9C	364 DFB \$9C		
F9AD:92	294 DFB \$92	:ABS,X		F9F2:6D	365 DFB \$6D		
F9AE:86	295 DFB \$86	:ABS,Y		F9F3:9C	366 DFB \$9C		
F9AF:8A	296 DFB \$8A	:ZPAG,Y		F9F4:8B	367 DFB \$8B		
F9B0:85	297 DFB \$85	:ZPAG,Y		F9F5:69	368 DFB \$69		
F9B1:9D	298 DFB \$9D	:RELATIVE		F9F6:29	369 DFB \$29		
F9B2:49	299 DFB \$49	:ZPAG (new)		F9F7:53	370 DFB \$53	: (C) FORMAT	
F9B3:5A	300 DFB \$5A	:ZPAG,X (new)		F9F8:84	371 DFB \$84		
F9B4:09	301 DFB \$09	:YV		F9F9:13	372 DFB \$13		
F9B5:08	302 DFB \$08	:YV		F9FA:14	373 DFB \$14		
F9B6:D8	303 DFB \$D8	:Byte F of FMT2		F9FB:11	374 DFB \$11		
F9B7:A4	304 DFB \$A4	:YV		F9FC:A5	375 DFB \$A5		
F9B8:A4	305 DFB \$A4	:YV		F9FD:69	376 DFB \$69		
F9B9:80	306 DFB \$80	:YV		F9FE:23	377 DFB \$23	: (E) FORMAT	
F9BA:AC	307 DFB \$AC	:YV		F9FF:A0	378 DFB \$A0		
F9BB:AC	308 DFB \$AC	:YV		F9A0:88	379 DFB \$88		
F9BC:A9	309 DFB \$A9	:YV		F9A1:62	380 DFB \$62		
F9BC:AC	310 DFB \$AC	:YV		F9A2:5A	381 DFB \$5A		
F9BD:A3	311 DFB \$A3	:YV		F9A3:4B	382 DFB \$4B		
F9BE:AB	312 DFB \$AB	:YV		F9A4:26	383 DFB \$26		
F9BF:1C	313 DFB \$1C	:YV		F9A5:94	384 DFB \$94		
F9C0:1C	314 DFB \$1C	:YV		F9A6:94	385 DFB \$94		
F9C1:8A	315 DFB \$8A	:YV		F9A7:8B	386 DFB \$8B		
F9C2:1C	316 DFB \$1C	:YV		F9A8:54	387 DFB \$54		
F9C3:23	317 DFB \$23	:YV		F9A9:44	388 DFB \$44		
F9C4:1B	318 DFB \$1B	:YV		F9AA:54	389 DFB \$54		
F9C5:8B	319 DFB \$8B	:YV		F9AB:54	390 DFB \$54		
F9C6:1B	320 DFB \$1B	:YV		F9AC:68	391 DFB \$68		
F9C7:A1	321 DFB \$A1	:YV		F9AD:44	392 DFB \$44		
F9C8:9D	322 DFB \$9D	:YV		F9AE:EB	393 DFB \$EB		
F9C9:9D	323 DFB \$9D	:YV		F9AF:EB	394 DFB \$EB		
F9CA:9D	324 DFB \$9D	:YV		F9B0:EB	395 DFB \$EB		
F9CB:23	325 DFB \$23	:YV		F9B1:EB	396 DFB \$EB		
F9CC:9D	326 DFB \$9D	:YV		F9B2:EB	397 DFB \$EB		
F9CD:8B	327 DFB \$8B	:YV		F9B3:EB	398 DFB \$EB		
F9CE:1D	328 DFB \$1D	:YV		F9B4:EB	399 DFB \$EB		
F9CF:1C	329 DFB \$1C	:YV		F9B5:EB	400 DFB \$EB		
F9D0:1C	330 DFB \$1C	:YV		F9B6:EB	401 DFB \$EB		
F9D1:29	331 DFB \$29	:YV		F9B7:EB	402 DFB \$EB		
F9D2:19	332 DFB \$19	:YV		F9B8:EB	403 DFB \$EB		
F9D3:AE	333 DFB \$AE	:YV		F9B9:EB	404 DFB \$EB		
F9D4:69	334 DFB \$69	:YV		F9BA:EB	405 DFB \$EB		
F9D5:19	335 DFB \$19	:YV		F9BB:EB	406 DFB \$EB		
F9D6:19	336 DFB \$19	:YV		F9BC:EB	407 DFB \$EB		
F9D7:23	337 DFB \$23	:YV		F9BD:EB	408 DFB \$EB		
F9D8:24	338 DFB \$24	:YV		F9BE:EB	409 DFB \$EB		
F9D9:53	339 DFB \$53	:YV		F9BF:EB	410 DFB \$EB		
F9DA:1B	340 DFB \$1B	:YV		F9C0:EB	411 DFB \$EB	: (A) FORMAT	
F9DB:23	341 DFB \$23	:YV		F9C1:EB	412 DFB \$EB	: TSB	
F9DC:23	342 DFB \$23	:YV		F9C2:EB	413 DFB \$EB		
F9DD:53	343 DFB \$53	:YV		F9C3:EB	414 DFB \$EB		
F9DE:19	344 DFB \$19	:YV		F9C4:EB	415 DFB \$EB		
F9DF:A1	345 DFB \$A1	:YV		F9C5:EB	416 DFB \$EB		
F9E0:AD	346 DFB \$AD	:YV		F9C6:EB	417 DFB \$EB		
F9E1:AD	347 DFB \$AD	:YV		F9C7:EB	418 DFB \$EB		
F9E2:5B	348 DFB \$5B	:YV		F9C8:EB	419 DFB \$EB		
F9E3:5B	349 DFB \$5B	:YV		F9C9:EB	420 DFB \$EB		
F9E4:A5	350 DFB \$A5	:YV		F9CA:EB	421 DFB \$EB		
F9E5:69	351 DFB \$69	:YV		F9CB:EB	422 DFB \$EB		
F9E6:24	352 DFB \$24	:YV		F9CC:EB	423 DFB \$EB		
F9E7:24	353 DFB \$24	:YV		F9CD:EB	424 DFB \$EB		
F9E8:AE	354 DFB \$AE	:YV		F9CE:EB	425 DFB \$EB		
F9E9:AE	355 DFB \$AE	:YV		F9CF:EB	426 DFB \$EB	: (C) FORMAT	
F9EA:AE	356 DFB \$AE	:YV		F9D0:EB	427 DFB \$EB		
F9EB:AE	357 DFB \$EB	:YV		F9D1:EB	428 DFB \$EB		

```

429 F831:1A DFB #1A
430 F832:26 DFB #26
431 F833:26 DFB #26
432 F834:72 DFB #72
433 F835:02 DFB #02
434 F836:02 DFB #02
435 F837:08 DFB #08
436 F838:04 DFB #04
437 F839:0A DFB #0A
438 F83A:26 DFB #26
439 F83B:05 DFB #05
440 F83C:44 DFB #44
441 F83D:44 DFB #44
442 F83E:A2 DFB #A2
443 F83F:C8 DFB #C8
444 F840:45 IRQ
445 F841:85 LDA #85
446 F842:45 LDA #45
447 F843:08 JMP
448 F844:08 JMP
449 F845:08 JMP
450 F846:08 JMP
451 F847:08 JMP
452 F848:08 JMP
453 F849:08 JMP
454 F84A:08 JMP
455 F84B:08 JMP
456 F84C:08 JMP
457 F84D:08 JMP
458 F84E:08 JMP
459 F84F:08 JMP
460 F850:08 JMP
461 F851:08 JMP
462 F852:08 JMP
463 F853:08 JMP
464 F854:08 JMP
465 F855:08 JMP
466 F856:08 JMP
467 F857:08 JMP
468 F858:08 JMP
469 F859:08 JMP
470 F85A:08 JMP
471 F85B:08 JMP
472 F85C:08 JMP
473 F85D:08 JMP
474 F85E:08 JMP
475 F85F:08 JMP
476 F860:08 JMP
477 F861:08 JMP
478 F862:08 JMP
479 F863:08 JMP
480 F864:08 JMP
481 F865:08 JMP
482 F866:08 JMP
483 F867:08 JMP
484 F868:08 JMP
485 F869:08 JMP
486 F86A:08 JMP
487 F86B:08 JMP
488 F86C:08 JMP
489 F86D:08 JMP
490 F86E:08 JMP
491 F86F:08 JMP
492 F870:08 JMP
493 F871:08 JMP
494 F872:08 JMP
495 F873:08 JMP
496 F874:08 JMP
497 F875:08 JMP
498 F876:08 JMP
499 F877:08 JMP
500 F878:08 JMP
501 F879:08 JMP
502 F87A:08 JMP
503 F87B:08 JMP
504 F87C:08 JMP
505 F87D:08 JMP
506 F87E:08 JMP
507 F87F:08 JMP
508 F880:08 JMP
509 F881:08 JMP
510 F882:08 JMP
511 F883:08 JMP
512 F884:08 JMP
513 F885:08 JMP
514 F886:08 JMP
515 F887:08 JMP
516 F888:08 JMP
517 F889:08 JMP
518 F88A:08 JMP
519 F88B:08 JMP
520 F88C:08 JMP
521 F88D:08 JMP
522 F88E:08 JMP
523 F88F:08 JMP
524 F890:08 JMP
525 F891:08 JMP
526 F892:08 JMP
527 F893:08 JMP
528 F894:08 JMP
529 F895:08 JMP
530 F896:08 JMP
531 F897:08 JMP
532 F898:08 JMP
533 F899:08 JMP
534 F89A:08 JMP
535 F89B:08 JMP
536 F89C:08 JMP
537 F89D:08 JMP
538 F89E:08 JMP
539 F89F:08 JMP
540 F8A0:08 JMP
541 F8A1:08 JMP
542 F8A2:08 JMP
543 F8A3:08 JMP
544 F8A4:08 JMP
545 F8A5:08 JMP
546 F8A6:08 JMP
547 F8A7:08 JMP
548 F8A8:08 JMP
549 F8A9:08 JMP
550 F8AA:08 JMP
551 F8AB:08 JMP
552 F8AC:08 JMP
553 F8AD:08 JMP
554 F8AE:08 JMP
555 F8AF:08 JMP
556 F8B0:08 JMP
557 F8B1:08 JMP
558 F8B2:08 JMP
559 F8B3:08 JMP
560 F8B4:08 JMP
561 F8B5:08 JMP
562 F8B6:08 JMP
563 F8B7:08 JMP
564 F8B8:08 JMP
565 F8B9:08 JMP
566 F8BA:08 JMP
567 F8BB:08 JMP
568 F8BC:08 JMP
569 F8BD:08 JMP
570 F8BE:08 JMP
571 F8BF:08 JMP
572 F8C0:08 JMP
573 F8C1:08 JMP
574 F8C2:08 JMP
575 F8C3:08 JMP
576 F8C4:08 JMP
577 F8C5:08 JMP
578 F8C6:08 JMP
579 F8C7:08 JMP
580 F8C8:08 JMP
581 F8C9:08 JMP
582 F8CA:08 JMP
583 F8CB:08 JMP
584 F8CC:08 JMP
585 F8CD:08 JMP
586 F8CE:08 JMP
587 F8CF:08 JMP
588 F8D0:08 JMP
589 F8D1:08 JMP
590 F8D2:08 JMP
591 F8D3:08 JMP
592 F8D4:08 JMP
593 F8D5:08 JMP
594 F8D6:08 JMP
595 F8D7:08 JMP
596 F8D8:08 JMP
597 F8D9:08 JMP
598 F8DA:08 JMP
599 F8DB:08 JMP
600 F8DC:08 JMP
601 F8DD:08 JMP
602 F8DE:08 JMP
603 F8DF:08 JMP
604 F8E0:08 JMP
605 F8E1:08 JMP
606 F8E2:08 JMP
607 F8E3:08 JMP
608 F8E4:08 JMP
609 F8E5:08 JMP
610 F8E6:08 JMP
611 F8E7:08 JMP
612 F8E8:08 JMP
613 F8E9:08 JMP
614 F8EA:08 JMP
615 F8EB:08 JMP
616 F8EC:08 JMP
617 F8ED:08 JMP
618 F8EE:08 JMP
619 F8EF:08 JMP
620 F8F0:08 JMP
621 F8F1:08 JMP
622 F8F2:08 JMP
623 F8F3:08 JMP
624 F8F4:08 JMP
625 F8F5:08 JMP
626 F8F6:08 JMP
627 F8F7:08 JMP
628 F8F8:08 JMP
629 F8F9:08 JMP
630 F8FA:08 JMP
631 F8FB:08 JMP
632 F8FC:08 JMP
633 F8FD:08 JMP
634 F8FE:08 JMP
635 F8FF:08 JMP

```

```

26-JUL-85          PAGE 63

; SETPG3 MUST RETURN X=0
; SET PIR H
; Display our banner...
; JUMP $C600

; read mouse paddle
; INIT COUNT
; COMPENSATE FOR 1ST COUNT
; COUNT Y-REG EVERY 12 USEC.

```

[illegible]

26-JUL-85 PAGE 64

28	AUTOST2	Apple //c F8 monitor firmware	26-JUL-85	PAGE 66
FC35:		144 *	145	* NEMOPS translates the opcode in the Y register
FC35:		145 *	146	* to a mnemonic table index and returns with Z=1.
FC35:		146 *	147	* if Y is not a new opcode, Z=0.
FC35:		147 *	148 *	* if Y is not a new opcode, Z=0.
FC35:98		149	NEMOPS	iget the opcode
FC36:A2	16	150	LDX	*NUMOPS
FC38:DD	FE	151	NEMOP1	check through new opcodes
FC3B:FF	43	152	OPDBL_X	idea's it match?
FC3C:FF	43	153	BEQ	GETINDEX
FC3D:1A		154	BEQ	GETINDEX
FC3E:1A		155	BPTX	idea's check next one
FC3F:68	F8	156	NEMOP1	idea's check next one
FC41:		157 *	RTS	!not found, exit with BNE
FC41:80		157 *	BRK	
FC42:		158 *		
FC42:80	19	159	CLREOP1	!ESC F IS CLR TO END OF PAGE
FC44:A5	25	160	CLREOP2	
FC46:A8	24	161	PHA	CV
FC47:80	24	162	VTABZ	!SAVE CURRENT LINE NO. ON STACK
FC4A:2E	FC	163	LDX	!CLEAR TO END ADDRESS
FC4B:A0	86	164	LDY	!CLEAR FROM H INDEX=0 FOR REST
FC4D:A0	86	165	PLA	!INCREMENT CURRENT LINE NO.
FC5B:1A		166	INC	A
FC5B:1A	23	167	CMPL	WHNDMT
FC5C:15	23	168	CMPL	WHNDMT
FC5D:15	C8	169	CMPL	WHNDMT
FC5E:15	C8	170	CMPL	WHNDMT
FC5F:15	C8	171 *	CMPL	WHNDMT
FC5F:15	C8	172 *	CMPL	WHNDMT
FC5F:15	C8	173 *	CMPL	WHNDMT
FC5F:15	C8	174 *	CMPL	WHNDMT
FC5F:15	C8	175 *	CMPL	WHNDMT
FC5F:15	C8	176 *	CMPL	WHNDMT
FC5F:15	C8	177 *	CMPL	WHNDMT
FC5F:15	C8	178 *	CMPL	WHNDMT
FC5F:15	C8	179 *	CMPL	WHNDMT
FC5F:15	C8	180 *	CMPL	WHNDMT
FC5F:15	C8	181 *	CMPL	WHNDMT
FC5F:15	C8	182 *	CMPL	WHNDMT
FC5F:15	C8	183 *	CMPL	WHNDMT
FC5F:15	C8	184 *	CMPL	WHNDMT
FC5F:15	C8	185 *	CMPL	WHNDMT
FC5F:15	C8	186 *	CMPL	WHNDMT
FC5F:15	C8	187 *	CMPL	WHNDMT
FC5F:15	C8	188 *	CMPL	WHNDMT
FC5F:15	C8	189 *	CMPL	WHNDMT
FC5F:15	C8	190 *	CMPL	WHNDMT
FC5F:15	C8	191 *	CMPL	WHNDMT
FC5F:15	C8	192 *	CMPL	WHNDMT
FC5F:15	C8	193 *	CMPL	WHNDMT
FC5F:15	C8	194 *	CMPL	WHNDMT
FC5F:15	C8	195 *	CMPL	WHNDMT
FC5F:15	C8	196 *	CMPL	WHNDMT
FC5F:15	C8	197 *	CMPL	WHNDMT
FC5F:15	C8	198 *	CMPL	WHNDMT
FC5F:15	C8	199 *	CMPL	WHNDMT
FC5F:15	C8	200 *	CMPL	WHNDMT
FC5F:15	C8	201 *	CMPL	WHNDMT
FC5F:15	C8	202 *	CMPL	WHNDMT
FC5F:15	C8	203 *	CMPL	WHNDMT
FC5F:15	C8	204 *	CMPL	WHNDMT
FC5F:15	C8	205 *	CMPL	WHNDMT
FC5F:15	C8	206 *	CMPL	WHNDMT
FC5F:15	C8	207 *	CMPL	WHNDMT
FC5F:15	C8	208 *	CMPL	WHNDMT
FC5F:15	C8	209 *	CMPL	WHNDMT
FC5F:15	C8	210 *	CMPL	WHNDMT
FC5F:15	C8	211 *	CMPL	WHNDMT
FC5F:15	C8	212 *	CMPL	WHNDMT
FC5F:15	C8	213 *	CMPL	WHNDMT
FC5F:15	C8	214 *	CMPL	WHNDMT
FC5F:15	C8	215 *	CMPL	WHNDMT
FC5F:15	C8	216 *	CMPL	WHNDMT
FC5F:15	C8	217 *	CMPL	WHNDMT
FC5F:15	C8	218 *	CMPL	WHNDMT
FC5F:15	C8	219 *	CMPL	WHNDMT
FC5F:15	C8	220 *	CMPL	WHNDMT


```

20 AUTOST2      215 *      BRA      NEMULEOLZ
FC02:00 EC      216 *      JMP      (CTLADR,X)
FC04:7C 2A CD      217 OTLDD
FC07:1E A      218 *      NOP
FC07:1E A      219 *      NOP
FC07:1E A      220 *      NOP
FC08:38      221 WAIT
FC09:48      222 WAIT2
FC0A:E9 01      223 WAIT3
FC0C:D0 FC      224 WAIT3
FC0E:E9 01      225 WAIT3
FC0F:10 00      226 WAIT3
FC0F:10 00      227 WAIT3
FC0F:10 00      228 RTS6
FC0F:10 00      229 *
FC0F:10 00      230 NHTA4
FC0F:10 00      231 NHTA4
FC0F:10 00      232 NHTA4
FC0F:10 00      233 NHTA1
FC0F:10 00      234 NHTA1
FC0F:10 00      235 NHTA1
FC0F:10 00      236 NHTA1
FC0F:10 00      237 NHTA1
FC0F:10 00      238 NHTA1
FC0F:10 00      239 NHTA1
FC0F:10 00      240 RTS4B
FC0F:10 00      241 *
FC0F:10 00      242 HEADR
FC0F:10 00      243 COLDDSTART
FC0F:10 00      244 COLDDSTART
FC0F:10 00      245 COLDDSTART
FC0F:10 00      246 COLDDSTART
FC0F:10 00      247 COLDDSTART
FC0F:10 00      248 COLDDSTART
FC0F:10 00      249 COLDDSTART
FC0F:10 00      250 COLDDSTART
FC0F:10 00      251 COLDDSTART
FC0F:10 00      252 COLDDSTART
FC0F:10 00      253 COLDDSTART
FC0F:10 00      254 COLDDSTART
FC0F:10 00      255 COLDDSTART
FC0F:10 00      256 COLDDSTART
FC0F:10 00      257 COLDDSTART
FC0F:10 00      258 COLDDSTART
FC0F:10 00      259 COLDDSTART
FC0F:10 00      260 COLDDSTART
FC0F:10 00      261 COLDDSTART
FC0F:10 00      262 COLDDSTART
FC0F:10 00      263 COLDDSTART
FC0F:10 00      264 COLDDSTART
FC0F:10 00      265 COLDDSTART
FC0F:10 00      266 COLDDSTART
FC0F:10 00      267 COLDDSTART
FC0F:10 00      268 COLDDSTART
FC0F:10 00      269 COLDDSTART
FC0F:10 00      270 COLDDSTART
FC0F:10 00      271 COLDDSTART
FC0F:10 00      272 COLDDSTART
FC0F:10 00      273 COLDDSTART
FC0F:10 00      274 COLDDSTART
FC0F:10 00      275 COLDDSTART
FC0F:10 00      276 COLDDSTART
FC0F:10 00      277 COLDDSTART
FC0F:10 00      278 COLDDSTART
FC0F:10 00      279 COLDDSTART
FC0F:10 00      280 COLDDSTART
FC0F:10 00      281 COLDDSTART
FC0F:10 00      282 COLDDSTART
FC0F:10 00      283 COLDDSTART
FC0F:10 00      284 COLDDSTART
FC0F:10 00      285 COLDDSTART

```

```

20 AUTOST2      286 COLDDSTART
FC0F:10 00      287 COLDDSTART
FC0F:10 00      288 COLDDSTART
FC0F:10 00      289 COLDDSTART
FC0F:10 00      290 COLDDSTART
FC0F:10 00      291 COLDDSTART
FC0F:10 00      292 COLDDSTART
FC0F:10 00      293 COLDDSTART
FC0F:10 00      294 COLDDSTART
FC0F:10 00      295 COLDDSTART
FC0F:10 00      296 COLDDSTART
FC0F:10 00      297 COLDDSTART
FC0F:10 00      298 COLDDSTART
FC0F:10 00      299 COLDDSTART
FC0F:10 00      300 COLDDSTART
FC0F:10 00      301 COLDDSTART
FC0F:10 00      302 COLDDSTART
FC0F:10 00      303 COLDDSTART
FC0F:10 00      304 COLDDSTART
FC0F:10 00      305 COLDDSTART
FC0F:10 00      306 COLDDSTART
FC0F:10 00      307 COLDDSTART
FC0F:10 00      308 COLDDSTART
FC0F:10 00      309 COLDDSTART
FC0F:10 00      310 COLDDSTART
FC0F:10 00      311 COLDDSTART
FC0F:10 00      312 COLDDSTART
FC0F:10 00      313 COLDDSTART
FC0F:10 00      314 COLDDSTART
FC0F:10 00      315 COLDDSTART
FC0F:10 00      316 COLDDSTART
FC0F:10 00      317 COLDDSTART
FC0F:10 00      318 COLDDSTART
FC0F:10 00      319 COLDDSTART
FC0F:10 00      320 COLDDSTART
FC0F:10 00      321 COLDDSTART
FC0F:10 00      322 COLDDSTART
FC0F:10 00      323 COLDDSTART
FC0F:10 00      324 COLDDSTART
FC0F:10 00      325 COLDDSTART
FC0F:10 00      326 COLDDSTART
FC0F:10 00      327 COLDDSTART
FC0F:10 00      328 COLDDSTART
FC0F:10 00      329 COLDDSTART
FC0F:10 00      330 COLDDSTART
FC0F:10 00      331 COLDDSTART
FC0F:10 00      332 COLDDSTART
FC0F:10 00      333 COLDDSTART
FC0F:10 00      334 COLDDSTART
FC0F:10 00      335 COLDDSTART
FC0F:10 00      336 COLDDSTART
FC0F:10 00      337 COLDDSTART
FC0F:10 00      338 COLDDSTART
FC0F:10 00      339 COLDDSTART
FC0F:10 00      340 COLDDSTART
FC0F:10 00      341 COLDDSTART
FC0F:10 00      342 COLDDSTART
FC0F:10 00      343 COLDDSTART
FC0F:10 00      344 COLDDSTART
FC0F:10 00      345 COLDDSTART
FC0F:10 00      346 COLDDSTART
FC0F:10 00      347 COLDDSTART
FC0F:10 00      348 COLDDSTART
FC0F:10 00      349 COLDDSTART
FC0F:10 00      350 COLDDSTART
FC0F:10 00      351 COLDDSTART
FC0F:10 00      352 COLDDSTART
FC0F:10 00      353 COLDDSTART
FC0F:10 00      354 COLDDSTART
FC0F:10 00      355 COLDDSTART
FC0F:10 00      356 COLDDSTART
FC0F:10 00      357 COLDDSTART
FC0F:10 00      358 COLDDSTART
FC0F:10 00      359 COLDDSTART
FC0F:10 00      360 COLDDSTART
FC0F:10 00      361 COLDDSTART
FC0F:10 00      362 COLDDSTART
FC0F:10 00      363 COLDDSTART
FC0F:10 00      364 COLDDSTART
FC0F:10 00      365 COLDDSTART
FC0F:10 00      366 COLDDSTART
FC0F:10 00      367 COLDDSTART
FC0F:10 00      368 COLDDSTART
FC0F:10 00      369 COLDDSTART
FC0F:10 00      370 COLDDSTART
FC0F:10 00      371 COLDDSTART
FC0F:10 00      372 COLDDSTART
FC0F:10 00      373 COLDDSTART
FC0F:10 00      374 COLDDSTART
FC0F:10 00      375 COLDDSTART
FC0F:10 00      376 COLDDSTART
FC0F:10 00      377 COLDDSTART
FC0F:10 00      378 COLDDSTART
FC0F:10 00      379 COLDDSTART
FC0F:10 00      380 COLDDSTART
FC0F:10 00      381 COLDDSTART
FC0F:10 00      382 COLDDSTART
FC0F:10 00      383 COLDDSTART
FC0F:10 00      384 COLDDSTART
FC0F:10 00      385 COLDDSTART
FC0F:10 00      386 COLDDSTART
FC0F:10 00      387 COLDDSTART
FC0F:10 00      388 COLDDSTART
FC0F:10 00      389 COLDDSTART
FC0F:10 00      390 COLDDSTART
FC0F:10 00      391 COLDDSTART
FC0F:10 00      392 COLDDSTART
FC0F:10 00      393 COLDDSTART
FC0F:10 00      394 COLDDSTART
FC0F:10 00      395 COLDDSTART
FC0F:10 00      396 COLDDSTART
FC0F:10 00      397 COLDDSTART
FC0F:10 00      398 COLDDSTART
FC0F:10 00      399 COLDDSTART
FC0F:10 00      400 COLDDSTART

```


20 AUTOST2	Apple //c F8 monitor firmware	26-JUL-85	PAGE 71
499 *	LDY #03F	LDY #03F	LDY #03F
500 SETINV	LDY #03F	LDY #03F	LDY #03F
FE06	LDY #03F	LDY #03F	LDY #03F
501 SETINV	LDY #03F	LDY #03F	LDY #03F
502 SETINV	LDY #03F	LDY #03F	LDY #03F
503 SETINV	LDY #03F	LDY #03F	LDY #03F
504	LDY #03F	LDY #03F	LDY #03F
FE07	LDY #03F	LDY #03F	LDY #03F
505	LDY #03F	LDY #03F	LDY #03F
FE08	LDY #03F	LDY #03F	LDY #03F
506	LDY #03F	LDY #03F	LDY #03F
FE09	LDY #03F	LDY #03F	LDY #03F
507	LDY #03F	LDY #03F	LDY #03F
FE10	LDY #03F	LDY #03F	LDY #03F
508	LDY #03F	LDY #03F	LDY #03F
FE11	LDY #03F	LDY #03F	LDY #03F
509	LDY #03F	LDY #03F	LDY #03F
FE12	LDY #03F	LDY #03F	LDY #03F
510	LDY #03F	LDY #03F	LDY #03F
FE13	LDY #03F	LDY #03F	LDY #03F
511	LDY #03F	LDY #03F	LDY #03F
FE14	LDY #03F	LDY #03F	LDY #03F
512	LDY #03F	LDY #03F	LDY #03F
FE15	LDY #03F	LDY #03F	LDY #03F
513	LDY #03F	LDY #03F	LDY #03F
FE16	LDY #03F	LDY #03F	LDY #03F
514	LDY #03F	LDY #03F	LDY #03F
FE17	LDY #03F	LDY #03F	LDY #03F
515	LDY #03F	LDY #03F	LDY #03F
FE18	LDY #03F	LDY #03F	LDY #03F
516	LDY #03F	LDY #03F	LDY #03F
FE19	LDY #03F	LDY #03F	LDY #03F
517	LDY #03F	LDY #03F	LDY #03F
FE20	LDY #03F	LDY #03F	LDY #03F
518	LDY #03F	LDY #03F	LDY #03F
FE21	LDY #03F	LDY #03F	LDY #03F
519	LDY #03F	LDY #03F	LDY #03F
FE22	LDY #03F	LDY #03F	LDY #03F
520	LDY #03F	LDY #03F	LDY #03F
FE23	LDY #03F	LDY #03F	LDY #03F
521	LDY #03F	LDY #03F	LDY #03F
FE24	LDY #03F	LDY #03F	LDY #03F
522	LDY #03F	LDY #03F	LDY #03F
FE25	LDY #03F	LDY #03F	LDY #03F
523	LDY #03F	LDY #03F	LDY #03F
FE26	LDY #03F	LDY #03F	LDY #03F
524	LDY #03F	LDY #03F	LDY #03F
FE27	LDY #03F	LDY #03F	LDY #03F
525	LDY #03F	LDY #03F	LDY #03F
FE28	LDY #03F	LDY #03F	LDY #03F
526	LDY #03F	LDY #03F	LDY #03F
FE29	LDY #03F	LDY #03F	LDY #03F
527	LDY #03F	LDY #03F	LDY #03F
FE30	LDY #03F	LDY #03F	LDY #03F
528	LDY #03F	LDY #03F	LDY #03F
FE31	LDY #03F	LDY #03F	LDY #03F
529	LDY #03F	LDY #03F	LDY #03F
FE32	LDY #03F	LDY #03F	LDY #03F
530	LDY #03F	LDY #03F	LDY #03F
FE33	LDY #03F	LDY #03F	LDY #03F
531	LDY #03F	LDY #03F	LDY #03F
FE34	LDY #03F	LDY #03F	LDY #03F
532	LDY #03F	LDY #03F	LDY #03F
FE35	LDY #03F	LDY #03F	LDY #03F
533	LDY #03F	LDY #03F	LDY #03F
FE36	LDY #03F	LDY #03F	LDY #03F
534	LDY #03F	LDY #03F	LDY #03F
FE37	LDY #03F	LDY #03F	LDY #03F
535	LDY #03F	LDY #03F	LDY #03F
FE38	LDY #03F	LDY #03F	LDY #03F
536	LDY #03F	LDY #03F	LDY #03F
FE39	LDY #03F	LDY #03F	LDY #03F
537	LDY #03F	LDY #03F	LDY #03F
FE40	LDY #03F	LDY #03F	LDY #03F
538	LDY #03F	LDY #03F	LDY #03F
FE41	LDY #03F	LDY #03F	LDY #03F
539	LDY #03F	LDY #03F	LDY #03F
FE42			

20 AUTOST2	Apple //c F8 monitor firmware	26-JUL-85	PAGE 71
499 *	LDY #03F	LDY #03F	LDY #03F
500 SETINV	LDY #03F	LDY #03F	LDY #03F
FE06	LDY #03F	LDY #03F	LDY #03F
501 SETINV	LDY #03F	LDY #03F	LDY #03F
502 SETINV	LDY #03F	LDY #03F	LDY #03F
503 SETINV	LDY #03F	LDY #03F	LDY #03F
504	LDY #03F	LDY #03F	LDY #03F
FE07	LDY #03F	LDY #03F	LDY #03F
505	LDY #03F	LDY #03F	LDY #03F
FE08	LDY #03F	LDY #03F	LDY #03F
506	LDY #03F	LDY #03F	LDY #03F
FE09	LDY #03F	LDY #03F	LDY #03F
507	LDY #03F	LDY #03F	LDY #03F
FE10	LDY #03F	LDY #03F	LDY #03F
508	LDY #03F	LDY #03F	LDY #03F
FE11	LDY #03F	LDY #03F	LDY #03F
509	LDY #03F	LDY #03F	LDY #03F
FE12	LDY #03F	LDY #03F	LDY #03F
510	LDY #03F	LDY #03F	LDY #03F
FE13	LDY #03F	LDY #03F	LDY #03F
511	LDY #03F	LDY #03F	LDY #03F
FE14	LDY #03F	LDY #03F	LDY #03F
512	LDY #03F	LDY #03F	LDY #03F
FE15	LDY #03F	LDY #03F	LDY #03F
513	LDY #03F	LDY #03F	LDY #03F
FE16	LDY #03F	LDY #03F	LDY #03F
514	LDY #03F	LDY #03F	LDY #03F
FE17	LDY #03F	LDY #03F	LDY #03F
515	LDY #03F	LDY #03F	LDY #03F
FE18	LDY #03F	LDY #03F	LDY #03F
516	LDY #03F	LDY #03F	LDY #03F
FE19	LDY #03F	LDY #03F	LDY #03F
517	LDY #03F	LDY #03F	LDY #03F
FE20	LDY #03F	LDY #03F	LDY #03F
518	LDY #03F	LDY #03F	LDY #03F
FE21	LDY #03F	LDY #03F	LDY #03F
519	LDY #03F	LDY #03F	LDY #03F
FE22	LDY #03F	LDY #03F	LDY #03F
520	LDY #03F	LDY #03F	LDY #03F
FE23	LDY #03F	LDY #03F	LDY #03F
521	LDY #03F	LDY #03F	LDY #03F
FE24	LDY #03F	LDY #03F	LDY #03F
522	LDY #03F	LDY #03F	LDY #03F
FE25	LDY #03F	LDY #03F	LDY #03F
523	LDY #03F	LDY #03F	LDY #03F
FE26	LDY #03F	LDY #03F	LDY #03F
524	LDY #03F	LDY #03F	LDY #03F
FE27	LDY #03F	LDY #03F	LDY #03F
525	LDY #03F	LDY #03F	LDY #03F
FE28	LDY #03F	LDY #03F	LDY #03F
526	LDY #03F	LDY #03F	LDY #03F
FE29	LDY #03F	LDY #03F	LDY #03F
527	LDY #03F	LDY #03F	LDY #03F
FE30	LDY #03F	LDY #03F	LDY #03F
528	LDY #03F	LDY #03F	LDY #03F
FE31	LDY #03F	LDY #03F	LDY #03F
529	LDY #03F	LDY #03F	LDY #03F
FE32	LDY #03F	LDY #03F	LDY #03F
530	LDY #03F	LDY #03F	LDY #03F
FE33	LDY #03F	LDY #03F	LDY #03F
531	LDY #03F	LDY #03F	LDY #03F
FE34	LDY #03F	LDY #03F	LDY #03F
532	LDY #03F	LDY #03F	LDY #03F
FE35	LDY #03F	LDY #03F	LDY #03F
533	LDY #03F	LDY #03F	LDY #03F
FE36	LDY #03F	LDY #03F	LDY #03F
534	LDY #03F	LDY #03F	LDY #03F
FE37	LDY #03F	LDY #03F	LDY #03F
535	LDY #03F	LDY #03F	LDY #03F
FE38	LDY #03F	LDY #03F	LDY #03F
536	LDY #03F	LDY #03F	LDY #03F
FE39	LDY #03F	LDY #03F	LDY #03F
537	LDY #03F	LDY #03F	LDY #03F
FE40	LDY #03F	LDY #03F	LDY #03F
538	LDY #03F	LDY #03F	LDY #03F
FE41	LDY #03F	LDY #03F	LDY #03F
539	LDY #03F	LDY #03F	LDY #03F
FE42			

20 AUTOST2	Apple //c F8 monitor firmware	26-JUL-85	PAGE 71
499 *	LDY #03F	LDY #03F	LDY #03F
500 SETINV	LDY #03F	LDY #03F	LDY #03F
FE06	LDY #03F	LDY #03F	LDY #03F
501 SETINV	LDY #03F	LDY #03F	LDY #03F
502 SETINV	LDY #03F	LDY #03F	LDY #03F
503 SETINV	LDY #03F	LDY #03F	LDY #03F
504	LDY #03F	LDY #03F	LDY #03F
FE07	LDY #03F	LDY #03F	LDY #03F
505	LDY #03F	LDY #03F	LDY #03F
FE08	LDY #03F	LDY #03F	LDY #03F
506	LDY #03F	LDY #03F	LDY #03F
FE09	LDY #03F	LDY #03F	LDY #03F
507	LDY #03F	LDY #03F	LDY #03F
FE10	LDY #03F	LDY #03F	LDY #03F
508	LDY #03F	LDY #03F	LDY #03F
FE11	LDY #03F	LDY #03F	LDY #03F
509	LDY #03F	LDY #03F	LDY #03F
FE12	LDY #03F	LDY #03F	LDY #03F
510	LDY #03F	LDY #03F	LDY #03F
FE13	LDY #03F	LDY #03F	LDY #03F
511	LDY #03F	LDY #03F	LDY #03F
FE14	LDY #03F	LDY #03F	LDY #03F
512	LDY #03F	LDY #03F	LDY #03F
FE15	LDY #03F	LDY #03F	LDY #03F
513	LDY #03F	LDY #03F	LDY #03F
FE16	LDY #03F	LDY #03F	LDY #03F
514	LDY #03F	LDY #03F	LDY #03F
FE17	LDY #03F	LDY #03F	LDY #03F
515	LDY #03F	LDY #03F	LDY #03F
FE18	LDY #03F	LDY #03F	LDY #03F
516	LDY #03F	LDY #03F	LDY #03F
FE19	LDY #03F	LDY #03F	LDY #03F
517	LDY #03F	LDY #03F	LDY #03F
FE20	LDY #03F	LDY #03F	LDY #03F
518	LDY #03F	LDY #03F	LDY #03F
FE21	LDY #03F	LDY #03F	LDY #03F
519	LDY #03F	LDY #03F	LDY #03F
FE22	LDY #03F	LDY #03F	LDY #03F
520	LDY #03F	LDY #03F	LDY #03F
FE23	LDY #03F	LDY #03F	LDY #03F
521	LDY #03F	LDY #03F	LDY #03F
FE24	LDY #03F	LDY #03F	LDY #03F
522	LDY #03F	LDY #03F	LDY #03F
FE25	LDY #03F	LDY #03F	LDY #03F
523	LDY #03F	LDY #03F	LDY #03F
FE26	LDY #03F	LDY #03F	LDY #03F
524	LDY #03F	LDY #03F	LDY #03F
FE27	LDY #03F	LDY #03F	LDY #03F
525	LDY #03F	LDY #03F	LDY #03F
FE28	LDY #03F	LDY #03F	LDY #03F
526	LDY #03F	LDY #03F	LDY #03F
FE29	LDY #03F	LDY #03F	LDY #03F
527	LDY #03F	LDY #03F	LDY #03F
FE30	LDY #03F	LDY #03F	LDY #03F
528	LDY #03F	LDY #03F	LDY #03F
FE31	LDY #03F	LDY #03F	LDY #03F
529	LDY #03F	LDY #03F	LDY #03F
FE32	LDY #03F	LDY #03F	LDY #03F
530	LDY #03F	LDY #03F	LDY #03F
FE33	LDY #03F	LDY #03F	LDY #03F
531	LDY #03F	LDY #03F	LDY #03F
FE34	LDY #03F	LDY #03F	LDY #03F
532	LDY #03F	LDY #03F	LDY #03F
FE35	LDY #03F	LDY #03F	LDY #03F
533	LDY #03F	LDY #03F	LDY #03F
FE36	LDY #03F	LDY #03F	LDY #03F
534	LDY #03F	LDY #03F	LDY #03F
FE37	LDY #03F	LDY #03F	LDY #03F
535	LDY #03F	LDY #03F	LDY #03F
FE38	LDY #03F	LDY #03F	LDY #03F
536	LDY #03F	LDY #03F	LDY #03F
FE39	LDY #03F	LDY #03F	LDY #03F
537	LDY #03F	LDY #03F	LDY #03F
FE40	LDY #03F	LDY #03F	LDY #03F
538	LDY #03F	LDY #03F	LDY #03F
FE41	LDY #03F	LDY #03F	LDY #03F
539	LDY #03F	LDY #03F	LDY #03F
FE42			

20 AUTOST2	Apple //c F8 monitor firmware	26-JUL-85	PAGE 71
499 *	LDY #03F	LDY #03F	LDY #03F
500 SETINV	LDY #03F	LDY #03F	LDY #03F
FE06	LDY #03F	LDY #03F	LDY #03F
501 SETINV	LDY #03F	LDY #03F	LDY #03F
502 SETINV	LDY #03F	LDY #03F	LDY #03F
503 SETINV	LDY #03F	LDY #03F	LDY #03F
504	LDY #03F	LDY #03F	LDY #03F
FE07	LDY #03F	LDY #03F	LDY #03F
505	LDY #03F	LDY #03F	LDY #03F
FE08	LDY #03F	LDY #03F	LDY #03F
506	LDY #03F	LDY #03F	LDY #03F
FE09	LDY #03F	LDY #03F	LDY #03F
507	LDY #03F	LDY #03F	LDY #03F
FE10	LDY #03F	LDY #03F	LDY #03F
508	LDY #03F	LDY #03F	LDY #03F
FE11	LDY #03F	LDY #03F	LDY #03F
509	LDY #03F	LDY #03F	LDY #03F
FE12	LDY #03F	LDY #03F	LDY #03F
510	LDY #03F	LDY #03F	LDY #03F
FE13	LDY #03F	LDY #03F	LDY #03F
511	LDY #03F	LDY #03F	LDY #03F
FE14	LDY #03F	LDY #03F	LDY #03F
512	LDY #03F	LDY #03F	LDY #03F
FE15	LDY #03F	LDY #03F	LDY #03F
513	LDY #03F	LDY #03F	LDY #03F
FE16	LDY #03F	LDY #03F	LDY #03F
514	LDY #03F	LDY #03F	LDY #03F
FE17	LDY #03F	LDY #03F	LDY #03F
515	LDY #03F	LDY #03F	LDY #03F
FE18	LDY #03F	LDY #03F	LDY #03F
516	LDY #03F	LDY #03F	LDY #03F
FE19	LDY #03F	LDY #03F	LDY #03F
517	LDY #03F	LDY #03F	LDY #03F
FE20	LDY #03F	LDY #03F	LDY #03F
518	LDY #03F	LDY #03F	LDY #03F
FE21	LDY #03F	LDY #03F	LDY #03F
519	LDY #03F	LDY #03F	LDY #03F
FE22	LDY #03F		

```

FF2D:A9 C5 641 PRERR LDA #6C5
FF2E:20 ED FD 642 JSR COUT
FF2F:20 ED FD 643 JSR COUT
FF30:20 ED FD 644 JSR COUT
FF31:20 ED FD 645 JSR COUT
FF3A: 646 *
FF3A:A9 87 LDA #687
FF3C:4C ED FD 648 JMP COUT
FF3E:15 48 FFE3 RESTORE LDA STATUS
FF41:14B 651 LDA #5H
FF42:A5 45 652 RESTR1 LDA XREG
FF44:A6 46 653 RESTR1 LDY XREG
FF46:A4 47 654 PLP
FF48:20 655 RTS
FF49:20 656 *
FF49:20 657 *
FF4A:95 45 658 SAVE STA XREG
FF4C:86 46 659 SAV1 STY XREG
FF4E:94 47 660 PHP
FF50:80 661 STA STATUS
FF52:85 48 663 STX SPNT
FF54:8A 664 STX SPNT
FF55:86 49 665 CLD
FF57:08 666 CLD
FF58:08 667 *
FF58:08 668 *
FF59:20 84 FE 669 OLDRST JSR INIT
FF5C:20 2F FB 670 JSR SETVID
FF5E:20 93 FE 671 JSR SETKBD
FF62:20 89 FE 672 CLD
FF63:08 673 *
FF63:08 674 *
FF63:08 675 *
FF63:20 3A FF 676 MONZ LDA #6AA
FF6B:85 33 677 STA PROMPT
FF6D:20 67 FD 678 JSR GETLNZ
FF6E:20 67 FD 679 JSR GETLNZ
FF70:20 47 FF 680 NXTITM JSR GETTWM
FF72:84 31 681 STY VSAV
FF73:A0 17 682 LDY #SUBTBL-CHRTBL
FF74:08 683 DEY
FF75:30 E8 684 BMT MON
FF77:D9 CC FF 685 CMP CHRTBL-Y
FF78:D0 F8 FF7A BNE CHRSRCH
FF7A:20 BE FF 687 JSR TOSUB
FF7B:44 34 688 LDY VSAV
FF7C:4C 73 FF 689 JMP NXTITM
FF7E:A2 83 690 *
FF7E:A2 83 691 DIG LDX #603
FF80:8A 692 ASL A
FF81:8A 693 ASL A
FF82:8A 694 ASL A
FF83:8A 695 ASL A
FF84:8A 696 NXTBIT ASL A
FF85:26 3F 697 ROL A2L
FF86:8A 698 ROL A2H
FF87:8A 699 DEX
FF88:8A 699 DEX
FF89:8A 699 DEX
FF8A:8A 699 DEX
FF8B:8A 699 DEX
FF8C:8A 699 DEX
FF8D:8A 699 DEX
FF8E:8A 699 DEX
FF8F:8A 699 DEX
FF90:8A 699 DEX
FF91:26 3E 697 ROL A2L
FF92:26 3F 698 ROL A2H
FF93:8A 699 DEX
FF94:8A 699 DEX
FF95:8A 699 DEX
FF96:8A 699 DEX
FF97:8A 699 DEX
FF98:8A 699 DEX
FF99:8A 699 DEX
FF9A:8A 699 DEX
FF9B:8A 699 DEX
FF9C:8A 699 DEX
FF9D:8A 699 DEX
FF9E:8A 699 DEX
FF9F:8A 699 DEX
FFA0:8A 699 DEX
FFA1:8A 699 DEX
FFA2:8A 699 DEX
FFA3:8A 699 DEX
FFA4:8A 699 DEX
FFA5:8A 699 DEX
FFA6:8A 699 DEX
FFA7:8A 699 DEX
FFA8:8A 699 DEX
FFA9:8A 699 DEX
FFAA:8A 699 DEX
FFAB:8A 699 DEX
FFAC:8A 699 DEX
FFAD:8A 699 DEX
FFAE:8A 699 DEX
FFAF:8A 699 DEX
FFB0:8A 699 DEX
FFB1:8A 699 DEX
FFB2:8A 699 DEX
FFB3:8A 699 DEX
FFB4:8A 699 DEX
FFB5:8A 699 DEX
FFB6:8A 699 DEX
FFB7:8A 699 DEX
FFB8:8A 699 DEX
FFB9:8A 699 DEX
FFBA:8A 699 DEX
FFBB:8A 699 DEX
FFBC:8A 699 DEX
FFBD:8A 699 DEX
FFBE:8A 699 DEX
FFBF:8A 699 DEX
FFC0:8A 699 DEX
FFC1:8A 699 DEX
FFC2:8A 699 DEX
FFC3:8A 699 DEX
FFC4:8A 699 DEX
FFC5:8A 699 DEX
FFC6:8A 699 DEX
FFC7:8A 699 DEX
FFC8:8A 699 DEX
FFC9:8A 699 DEX
FFCA:8A 699 DEX
FFCB:8A 699 DEX
FFCC:8A 699 DEX
FFCD:8A 699 DEX
FFCE:8A 699 DEX
FFCF:8A 699 DEX
FFD0:8A 699 DEX
FFD1:8A 699 DEX
FFD2:8A 699 DEX
FFD3:8A 699 DEX
FFD4:8A 699 DEX
FFD5:8A 699 DEX
FFD6:8A 699 DEX
FFD7:8A 699 DEX
FFD8:8A 699 DEX
FFD9:8A 699 DEX
FFDA:8A 699 DEX
FFDB:8A 699 DEX
FFDC:8A 699 DEX
FFDD:8A 699 DEX
FFDE:8A 699 DEX
FFDF:8A 699 DEX
FFE0:8A 699 DEX
FFE1:8A 699 DEX
FFE2:8A 699 DEX
FFE3:8A 699 DEX
FFE4:8A 699 DEX
FFE5:8A 699 DEX
FFE6:8A 699 DEX
FFE7:8A 699 DEX
FFE8:8A 699 DEX
FFE9:8A 699 DEX
FFEA:8A 699 DEX
FFEB:8A 699 DEX
FFEC:8A 699 DEX
FFED:8A 699 DEX
FFEE:8A 699 DEX
FFEF:8A 699 DEX

```

```

FFAD:20 B4 C5 712 NATCHR JSR GETUP
FFB0:49 80 713 COR #800
FFB1:49 80 714 COR #800
FFB2:49 80 715 BCC DIG
FFB3:49 80 716 BCC DIG
FFB4:49 80 717 BCC DIG
FFB5:49 80 718 BCC DIG
FFB6:49 80 719 BCC DIG
FFB7:49 80 720 BCC DIG
FFB8:49 80 721 BCC DIG
FFB9:49 80 722 BCC DIG
FFBA:49 80 723 BCC DIG
FFBB:49 80 724 BCC DIG
FFBC:49 80 725 BCC DIG
FFBD:49 80 726 BCC DIG
FFBE:49 80 727 BCC DIG
FFBF:49 80 728 BCC DIG
FFC0:49 80 729 BCC DIG
FFC1:49 80 730 BCC DIG
FFC2:49 80 731 BCC DIG
FFC3:49 80 732 BCC DIG
FFC4:49 80 733 BCC DIG
FFC5:49 80 734 BCC DIG
FFC6:49 80 735 BCC DIG
FFC7:49 80 736 BCC DIG
FFC8:49 80 737 BCC DIG
FFC9:49 80 738 BCC DIG
FFCA:49 80 739 BCC DIG
FFCB:49 80 740 BCC DIG
FFCC:49 80 741 BCC DIG
FFCD:49 80 742 BCC DIG
FFCE:49 80 743 BCC DIG
FFCF:49 80 744 BCC DIG
FFD0:49 80 745 BCC DIG
FFD1:49 80 746 BCC DIG
FFD2:49 80 747 BCC DIG
FFD3:49 80 748 BCC DIG
FFD4:49 80 749 BCC DIG
FFD5:49 80 750 BCC DIG
FFD6:49 80 751 BCC DIG
FFD7:49 80 752 BCC DIG
FFD8:49 80 753 BCC DIG
FFD9:49 80 754 BCC DIG
FFDA:49 80 755 BCC DIG
FFDB:49 80 756 BCC DIG
FFDC:49 80 757 BCC DIG
FFDD:49 80 758 BCC DIG
FFDE:49 80 759 BCC DIG
FFDF:49 80 760 BCC DIG
FFE0:49 80 761 BCC DIG
FFE1:49 80 762 BCC DIG
FFE2:49 80 763 BCC DIG
FFE3:49 80 764 BCC DIG
FFE4:49 80 765 BCC DIG
FFE5:49 80 766 BCC DIG
FFE6:49 80 767 BCC DIG
FFE7:49 80 768 BCC DIG
FFE8:49 80 769 BCC DIG
FFE9:49 80 770 BCC DIG
FFEA:49 80 771 BCC DIG
FFEB:49 80 772 BCC DIG
FFEC:49 80 773 BCC DIG
FFED:49 80 774 BCC DIG
FFEE:49 80 775 BCC DIG
FFEF:49 80 776 BCC DIG
FFFA:49 80 777 BCC DIG
FFFB:49 80 778 BCC DIG
FFFC:49 80 779 BCC DIG
FFFD:49 80 780 BCC DIG
FFFE:49 80 781 BCC DIG
FFFF:49 80 782 BCC DIG

```

20 AUTOST2
FFC:62 FA
FFE:03 C8
0000:

Apple //c F8 monitor firmware
783 DW RESET
784 IRQVECT DW NEWIRQ
50 include bank2

26-JUL-85 PAGE 75
:RESET VECTOR
:INTERRUPT REQUEST VECTOR

21 BANK2 Apple //c F8 monitor firmware 26-JUL-85 PAGE 76

0000: 2
0000: 3 *
0000: 4 * Bank 2 of the roms
0000: 5
0000: 6

--- NEXT OBJECT FILE NAME IS FIRM.2

C000: C000 7 org \$C000
C000: 0100 51 include mint
C000: 0100 1 ds \$C100-*.0

;Mouse & acia interrupt handler

[illegible]

C105:38
C106:
C106:AD 19 C0

```

96 * This routine will determine if the source of
97 * is either of the built in ACIAs. If neither port
98 * generated an interrupt, or the interrupt was due
99 * to an external source, the carry is set and
100 * 'unbuffered' receiver full, the carry is set and
101 * indicating an externally serviced interrupt.
102 * If the interrupt source was keyboard, 'buffered'
103 * serial input, or the DSD, the interrupt is serviced
104 * and the carry is cleared. If the interrupt was
105 * serviced, C0BD handshake replaces C1S1 interrupt was
106 * serviced.
107 * Location "ACIABUF" specifies which (if either) re-
108 * ceiver data is buffered. For port 1 it must contain
109 * $C01, for port 2 $C02. Any other values are cause
110 * an interrupt to be passed to external RAM based routines.
111 * Interrupts are passed to external RAM based routines.
112 * RAM should be buffered ignored, or processed by
113 * RAM based routines. If bit 7-1 and bit 6-0, key-
114 * board data is placed in the type-ahead buffer. If
115 * bit 6 is set the interrupt is cleared, but must
116 * be recognized and serviced by a RAM routine. If the
117 * keyboard data is ignored, interrupt is serviced, but the
118 * while using type-ahead, Open-Apple CTRL-X will
119 * flush the buffer. No other code is recognized.
120 * If the source was an ACIA that has the transmit
121 * interrupt enabled, the original value of the ACIAs
122 * buffer register is preserved. Automatic acknowledgment
123 * buffer register is preserved. Automatic acknowledgment
124 * interrupt originating from the protocol converter or
125 * keyboard (RAM serviced) do not inhibit serial buffering
126 * and are passed thru. The RAM service routine can re-
127 * ognize the interrupt source by a 1 state in bit 6 of
128 * the ACIA status register. The RAM service routine must
129 * pass a message to the status register. Second ac-
130 * cess to the status register before returning.
131 *
132 *
133 *
134 *
135 *
136 *
137 *
138 *
139 *
140 *
141 *
142 *
143 *
144 *
145 *
146 *
147 *
148 *
149 *
150 *
151 *
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
8
```

```

C24F: 215 *****
C24F: 216 *
C24F: 217 * SEROUT3 - Outputs a character to a cila
C24F: 218 * Inputs: A = char, X = Cn
C24F: 219 *****
C24F: 220 *****
C24F: 221 serout3 equ *
C24F: 222
C24F: 223 jmp serout4
C24F: 224 equ swrta2
C24F: 225
C24F: 226 bnt sorta
C24F: 227 bne sordy
C24F: 228 unq col,x
C24F: 229 sordy
C24F: 230 and #18
C24F: 231 cmp #18
C24F: 232 bne sordy
C24F: 233 ldr #18,x
C24F: 234 bnt sorta
C24F: 235 beq aclebuf
C24F: 236 cpz aclebuf
C24F: 237 beq sortat
C24F: 238 jmp xrdnobuf
C24F: 239 ldr #18,x
C24F: 240 ldy charbuf,x
C24F: 241 ldy charbuf,y
C24F: 242 lda flags,x
C24F: 243 ora #80
C24F: 244 lda flags,x
C24F: 245 and #80
C24F: 246 stotat
C24F: 247 bne sordy
C24F: 248 ldy devno2,x
C24F: 249 pla
C24F: 250 stotat
C24F: 251 sta sdata,y
C24F: 252 bit flags,x
C24F: 253 eor #80
C24F: 254 asl A
C24F: 255 bne sodone
C24F: 256 bne sortat
C24F: 257 lda #114
C24F: 258 ror A
C24F: 259 jmp serout4
C24F: 260 clrcol
C24F: 261 siz ch
C24F: 262 bne sodone
C24F: 263 sorta
C24F: 264 *****
C24F: 265 *
C24F: 266 *
C24F: 267 * GETSTAT - Gets the status from a cila
C24F: 268 * GETSTAT2 - Call from this side
C24F: 269 * If interrupt, acistat is called
C24F: 270 * note: external interrupts are lost
C24F: 271 * Inputs: X = Cn
C24F: 272 * Outputs: A = status, X = Cn, Y = devno
C24F: 273 *****
C24F: 274 *****
C24F: 275 getstat equ *
C24F: 276 jmp getstat2
C24F: 277 equ swrta2
C24F: 278 getstat2
C24F: 279 php
C24F: 280 sei
C24F: 281 ldy devno2,x
C24F: 282 ldy getstat
C24F: 283 ldy acistat
C24F: 284 ldy gatno2
C24F: 285 *****

```

```

C280:28 D5 C1
C28F:08 F3 C284
C2C1:28
C2C2:60

```

284
285
286
287

Mouse & serial interrupt stuff

```

:Go service the interrupt
:Interrupt may have changed status
:Restore interrupt status

```

22 MINT

[illegible]

```

C322:
C323:
C324:
C325:
C326:
C327:
C328:
C329:
C330:
C331:
C332:
C333:
C334:
C335:
C336:
C337:
C338:
C339:
C340:
C341:
C342:
C343:
C344:
C345:
C346:
C347:
C348:
C349:
C350:
C351:
C352:
C353:
C354:
C355:
C356:
C357:
C358:
C359:
C360:
C361:
C362:
C363:
C364:
C365:
C366:
C367:
C368:
C369:
C370:
C371:
C372:
C373:
C374:
C375:
C376:
C377:
C378:
C379:
C380:
C381:
C382:
C383:
C384:
C385:
C386:
C387:
C388:
C389:
C390:
C391:
C392:
C393:
C394:
C395:
C396:
C397:
C398:
C399:
C400:
C401:
C402:
C403:
C404:
C405:
C406:
C407:
C408:
C409:
C410:
C411:
C412:
C413:
C414:
C415:
C416:
C417:
C418:
C419:
C420:
C421:
C422:
C423:
C424:
C425:
C426:
C427:
C428:
C429:
C430:
C431:
C432:
C433:
C434:
C435:
C436:
C437:
C438:
C439:
C440:
C441:
C442:
C443:
C444:
C445:
C446:
C447:
C448:
C449:
C450:
C451:
C452:
C453:
C454:
C455:
C456:
C457:
C458:
C459:
C460:
C461:
C462:
C463:
C464:
C465:
C466:
C467:
C468:
C469:
C470:
C471:
C472:
C473:
C474:
C475:
C476:
C477:
C478:
C479:
C480:
C481:
C482:
C483:
C484:
C485:
C486:
C487:
C488:
C489:
C490:
C491:
C492:
C493:
C494:
C495:
C496:
C497:
C498:
C499:
C500:
C501:
C502:
C503:
C504:
C505:
C506:
C507:
C508:
C509:
C510:
C511:
C512:
C513:
C514:
C515:
C516:
C517:
C518:
C519:
C520:
C521:
C522:
C523:
C524:
C525:
C526:
C527:
C528:
C529:
C530:
C531:
C532:
C533:
C534:
C535:
C536:
C537:
C538:
C539:
C540:
C541:
C542:
C543:
C544:
C545:
C546:
C547:
C548:
C549:
C550:
C551:
C552:
C553:
C554:
C555:
C556:
C557:
C558:
C559:
C560:
C561:
C562:
C563:
C564:
C565:
C566:
C567:
C568:
C569:
C570:
C571:
C572:
C573:
C574:
C575:
C576:
C577:
C578:
C579:
C580:
C581:
C582:
C583:
C584:
C585:
C586:
C587:
C588:
C589:
C590:
C591:
C592:
C593:
C594:
C595:
C596:
C597:
C598:
C599:
C600:
C601:
C602:
C603:
C604:
C605:
C606:
C607:
C608:
C609:
C610:
C611:
C612:
C613:
C614:
C615:
C616:
C617:
C618:
C619:
C620:
C621:
C622:
C623:
C624:
C625:
C626:
C627:
C628:
C629:
C630:
C631:
C632:
C633:
C634:
C635:
C636:
C637:
C638:
C639:
C640:
C641:
C642:
C643:
C644:
C645:
C646:
C647:
C648:
C649:
C650:
C651:
C652:
C653:
C654:
C655:
C656:
C657:
C658:
C659:
C660:
C661:
C662:
C663:
C664:
C665:
C666:
C667:
C668:
C669:
C670:
C671:
C672:
C673:
C674:
C675:
C676:
C677:
C678:
C679:
C680:
C681:
C682:
C683:
C684:
C685:
C686:
C687:
C688:
C689:
C690:
C691:
C692:
C693:
C694:
C695:
C696:
C697:
C698:
C699:
C700:
C701:
C702:
C703:
C704:
C705:
C706:
C707:
C708:
C709:
C710:
C711:
C712:
C713:
C714:
C715:
C716:
C717:
C718:
C719:
C720:
C721:
C722:
C723:
C724:
C725:
C726:
C727:
C728:
C729:
C730:
C731:
C732:
C733:
C734:
C735:
C736:
C737:
C738:
C739:
C740:
C741:
C742:
C743:
C744:
C745:
C746:
C747:
C748:
C749:
C750:
C751:
C752:
C753:
C754:
C755:
C756:
C757:
C758:
C759:
C760:
C761:
C762:
C763:
C764:
C765:
C766:
C767:
C768:
C769:
C770:
C771:
C772:
C773:
C774:
C775:
C776:
C777:
C778:
C779:
C780:
C781:
C782:
C783:
C784:
C785:
C786:
C787:
C788:
C789:
C790:
C791:
C792:
C793:
C794:
C795:
C796:
C797:
C798:
C799:
C800:
C801:
C802:
C803:
C804:
C805:
C806:
C807:
C808:
C809:
C810:
C811:
C812:
C813:
C814:
C815:
C816:
C817:
C818:
C819:
C820:
C821:
C822:
C823:
C824:
C825:
C826:
C827:
C828:
C829:
C830:
C831:
C832:
C833:
C834:
C835:
C836:
C837:
C838:
C839:
C840:
C841:
C842:
C843:
C844:
C845:
C846:
C847:
C848:
C849:
C850:
C851:
C852:
C853:
C854:
C855:
C856:
C857:
C858:
C859:
C860:
C861:
C862:
C863:
C864:
C865:
C866:
C867:
C868:
C869:
C870:
C871:
C872:
C873:
C874:
C875:
C876:
C877:
C878:
C879:
C880:
C881:
C882:
C883:
C884:
C885:
C886:
C887:
C888:
C889:
C890:
C891:
C892:
C893:
C894:
C895:
C896:
C897:
C898:
C899:
C900:
C901:
C902:
C903:
C904:
C905:
C906:
C907:
C908:
C909:
C910:
C911:
C912:
C913:
C914:
C915:
C916:
C917:
C918:
C919:
C920:
C921:
C922:
C923:
C924:
C925:
C926:
C927:
C928:
C929:
C930:
C931:
C932:
C933:
C934:
C935:
C936:
C937:
C938:
C939:
C940:
C941:
C942:
C943:
C944:
C945:
C946:
C947:
C948:
C949:
C950:
C951:
C952:
C953:
C954:
C955:
C956:
C957:
C958:
C959:
C960:
C961:
C962:
C963:
C964:
C965:
C966:
C967:
C968:
C969:
C970:
C971:
C972:
C973:
C974:
C975:
C976:
C977:
C978:
C979:
C980:
C981:
C982:
C983:
C984:
C985:
C986:
C987:
C988:
C989:
C990:
C991:
C992:
C993:
C994:
C995:
C996:
C997:
C998:
C999:

```

[illegible]

[illegible][illegible]

24 BANGER2	Apple //c diagnostics	26-JUL-85	PAGE 87
C366:	3 *****		
C366:	4 * Here is the rest of the diagnostic stuff		
C366:	5 * the first part of the diagnostic stuff		
C366:	6 * to mate desperately needed room		
C366:	8 *		
C366:	9 *****		
C366:	10 TSTMEM		
C366:	11 equ \$01		
C366:	12 stx \$02		
C366:	13 stx \$03		
C366:	14 ldx \$4		
C366:	15 ldx \$4		
C366:	16 MEM1		
C366:	17 stx \$05		
C366:	18 stx \$06		
C366:	19 inc 1		
C366:	20 mem2		
C366:	21 tay		
C366:	22 lcbank2		
C366:	23 stx \$02		
C366:	24 and \$FF		
C366:	25 cnp \$C0		
C366:	26 bne mem3		
C366:	27 ldx \$01		
C366:	28 ldx \$FF		
C366:	29 add mem4		
C366:	30 bne mem4		
C366:	31 ldx \$01		
C366:	32 mem3		
C366:	33 mem4		
C366:	34 tdx \$03		
C366:	35 ldy \$00		
C366:	36 mem5		
C366:	37 cnp		
C366:	38 stx \$02		
C366:	39 dnx		
C366:	40 ldx \$4		
C366:	41 ldx \$4		
C366:	42 mem6		
C366:	43 bne mem5		
C366:	44 inc 1		
C366:	45 bne mem2		
C366:	47 inc \$01		
C366:	48 LDX \$4		
C366:	49 LDA \$05		
C366:	50 mem7		
C366:	51 lcbank2		
C366:	52 ldx \$01		
C366:	53 ldx \$FF		
C366:	54 and \$C0		
C366:	55 cnp		
C366:	56 bne mem8		
C366:	57 ldx \$01		
C366:	58 ldx \$FF		
C366:	59 add mem9		
C366:	60 bne mem9		
C366:	61 ldx \$01		
C366:	62 mem9		
C366:	63 stx \$03		
C366:	64 ldy \$00		
C366:	65 memA		
C366:	66 cnp		
C366:	67 stx \$02		
C366:	68 bne		
C366:	69 ldx \$01		
C366:	70 dex		
C366:	71 bpl		
C366:	72 ldx \$4		
C366:	73 memB		
C366:	74 iny		

Apple //c diagnostics	26-JUL-85	PAGE 87
3 *****		
C306: 4 *****		
C306: 5 Here is the rest of the diagnostic stuff		
C306: 6 the first part has been moved into the \$D000 space		
C306: 7 to make desperately needed room		
C306: 8 *****		
C306: 9 *****		
C306: 10 *****		
C306: 11 TSTMEM equ \$01		
C306: 12 stx \$02		
C306: 13 stx \$03		
C306: 14 ldx \$4		
C306: 15 stx \$04		
C306: 16 stx \$05		
C306: 17 MEM1		
C306: 18 stx \$01		
C306: 19 inc 1		
C306: 20 mem2		
C306: 21 stx lcbank2		
C306: 22 stx lcbank2		
C306: 23 ldx \$04		
C306: 24 and \$08		
C306: 25 cmp \$08		
C306: 26 bne mem3		
C306: 27 ldx lcbank1		
C306: 28 ldx \$04		
C306: 29 cmp \$0F		
C306: 30 bne mem4		
C306: 31 bne mem4		
C306: 32 stx \$01		
C306: 33 stx \$03		
C306: 34 ldx \$08		
C306: 35 cbc mem5		
C306: 36 stx nblx		
C306: 37 adc (\$02),y		
C306: 38 dex		
C306: 39 ldx mem6		
C306: 40 ldx \$4		
C306: 41 bne mem5		
C306: 42 mem6		
C306: 43 C3FA		
C306: 44 inc 1		
C306: 45 inc 1		
C306: 46 inc 1		
C306: 47 inc \$01		
C306: 48 ldx \$4		
C306: 49 LDA \$05		
C306: 50 mem7		
C306: 51 ldx lcbank2		
C306: 52 ldx lcbank2		
C306: 53 ldx \$04		
C306: 54 cmp \$08		
C306: 55 cmp \$08		
C306: 56 bne mem8		
C306: 57 ldx lcbank1		
C306: 58 ldx \$0F		
C306: 59 ldx mem9		
C306: 60 bne mem9		
C306: 61 stx \$01		
C306: 62 stx \$03		
C306: 63 tye		
C306: 64 ldy \$08		
C306: 65 cmp \$08		
C306: 66 memA		
C306: 67 adc		
C306: 68 stx nblx		
C306: 69 (\$02),y		
C306: 70 bne MEMERRDR		
C306: 71 ldx		
C306: 72 bpl memB		
C306: 73 ldx \$4		
C306: 74 memB		
C306: 75 inc 1		
C306: 76 memB		
C306: 77 memB		
C306: 78 memB		
C306: 79 memB		
C306: 80 memB		
C306: 81 memB		
C306: 82 memB		
C306: 83 memB		
C306: 84 memB		
C306: 85 memB		
C306: 86 memB		
C306: 87 memB		
C306: 88 memB		
C306: 89 memB		
C306: 90 memB		
C306: 91 memB		
C306: 92 memB		
C306: 93 memB		
C306: 94 memB		
C306: 95 memB		
C306: 96 memB		
C306: 97 memB		
C306: 98 memB		
C306: 99 memB		
C306: 100 memB		
C306: 101 memB		
C306: 102 memB		
C306: 103 memB		
C306: 104 memB		
C306: 105 memB		
C306: 106 memB		
C306: 107 memB		
C306: 108 memB		
C306: 109 memB		
C306: 110 memB		
C306: 111 memB		
C306: 112 memB		
C306: 113 memB		
C306: 114 memB		
C306: 115 memB		
C306: 116 memB		
C306: 117 memB		
C306: 118 memB		
C306: 119 memB		
C306: 120 memB		
C306: 121 memB		
C306: 122 memB		
C306: 123 memB		
C306: 124 memB		
C306: 125 memB		
C306: 126 memB		
C306: 127 memB		
C306: 128 memB		
C306: 129 memB		
C306: 130 memB		
C306: 131 memB		
C306: 132 memB		
C306: 133 memB		
C306: 134 memB		
C306: 135 memB		
C306: 136 memB		
C306: 137 memB		
C306: 138 memB		
C306: 139 memB		
C306: 140 memB		
C306: 141 memB		
C306: 142 memB		
C306: 143 memB		
C306: 144 memB		
C306: 145 memB		
C306: 146 memB		
C306: 147 memB		
C306: 148 memB		
C306: 149 memB		
C306: 150 memB		
C306: 151 memB		
C306: 152 memB		
C306: 153 memB		
C306: 154 memB		
C306: 155 memB		
C306: 156 memB		
C306: 157 memB		
C306: 158 memB		
C306: 159 memB		
C306: 160 memB		
C306: 161 memB		
C306: 162 memB		
C306: 163 memB		
C306: 164 memB		
C306: 165 memB		
C306: 166 memB		
C306: 167 memB		
C306: 168 memB		
C306: 169 memB		
C306: 170 memB		
C306: 171 memB		
C306: 172 memB		
C306: 173 memB		
C306: 174 memB		
C306: 175 memB		
C306: 176 memB		
C306: 177 memB		
C306: 178 memB		
C306: 179 memB		
C306: 180 memB		
C306: 181 memB		
C306: 182 memB		
C306: 183 memB		
C306: 184 memB		
C306: 185 memB		
C306: 186 memB		
C306: 187 memB		
C306: 188 memB		
C306: 189 memB		
C306: 190 memB		
C306: 191 memB		
C306: 192 memB		
C306: 193 memB		
C306: 194 memB		
C306: 195 memB		
C306: 196 memB		
C306: 197 memB		
C306: 198 memB		
C306: 199 memB		
C306: 200 memB		
C306: 201 memB		
C306: 202 memB		
C306: 203 memB		
C306: 204 memB		
C306: 205 memB		
C306: 206 memB		
C306: 207 memB		
C306: 208 memB		
C306: 209 memB		
C306: 210 memB		
C306: 211 memB		
C306: 212 memB		
C306: 213 memB		
C306: 214 memB		
C306: 215 memB		
C306: 216 memB		
C306: 217 memB		
C306: 218 memB		
C306: 219 memB		
C306: 220 memB		
C306: 221 memB		
C306: 222 memB		
C306: 223 memB		
C306: 224 memB		
C306: 225 memB		
C306: 226 memB		
C306: 227 memB		
C306: 228 memB		
C306: 229 memB		
C306: 230 memB		
C306: 231 memB		
C306: 232 memB		
C306: 233 memB		
C306: 234 memB		
C306: 235 memB		
C306: 236 memB		
C306: 237 memB		
C306: 238 memB		
C306: 239 memB		
C306: 240 memB		
C306: 241 memB		
C306: 242 memB		
C306: 243 memB		
C306: 244 memB		
C306: 245 memB		
C306: 246 memB		
C306: 247 memB		
C306: 248 memB		
C306: 249 memB		
C306: 250 memB		
C306: 251 memB		
C306: 252 memB		
C306: 253 memB		
C306: 254 memB		
C306: 255 memB		
C306: 256 memB		
C306: 257 memB		
C306: 258 memB		
C306: 259 memB		
C306: 260 memB		
C306: 261 memB		
C306: 262 memB		
C306: 263 memB		
C306: 264 memB		
C306: 265 memB		
C306: 266 memB		
C306: 267 memB		
C306: 268 memB		
C306: 269 memB		
C306: 270 memB		
C306: 271 memB		
C306: 272 memB		
C306: 273 memB		
C306: 274 memB		
C306: 275 memB		
C306: 276 memB		
C306: 277 memB		
C306: 278 memB		
C306: 279 memB		
C306: 280 memB		
C306: 281 memB		
C306: 282 memB		
C306: 283 memB		
C306: 284 memB		
C306: 285 memB		
C306: 286 memB		
C306: 287 memB		
C306: 288 memB		
C306: 289 memB		
C306: 290 memB		
C306: 291 memB		
C306: 292 memB		
C306: 293 memB		
C306: 294 memB		
C306: 295 memB		
C306: 296 memB		
C306: 297 memB		
C306: 298 memB		
C306: 299 memB		
C306: 300 memB		
C306: 301 memB		
C306: 302 memB		
C306: 303 memB		
C306: 304 memB		
C306: 305 memB		
C306: 306 memB		
C306: 307 memB		
C306: 308 memB		
C306: 309 memB		
C306: 310 memB		
C306: 311 memB		
C306: 312 memB		
C306: 313 memB		
C306: 314 memB		
C306: 315 memB		
C306: 316 memB		
C306: 317 memB		
C306: 318 memB		
C306: 319 memB		
C306: 320 memB		
C306: 321 memB		
C306: 322 memB		
C306: 323 memB		
C306: 324 memB		
C306: 325 memB		
C306: 326 memB		
C306: 327 memB		
C306: 328 memB		
C306: 329 memB		
C306: 330 memB		
C306: 331 memB		
C306: 332 memB		
C306: 333 memB		
C306: 334 memB		
C306: 335 memB		
C306: 336 memB		
C306: 337 memB		
C306: 338 memB		
C306: 339 memB		
C306: 340 memB		
C306: 341 memB		
C306: 342 memB		
C306: 343 memB		
C306: 344 memB		
C306: 345 memB		
C306: 346 memB		
C306: 347 memB		
C306: 348 memB		
C306: 349 memB		
C306: 350 memB		
C306: 351 memB		
C306: 352 memB		
C306: 353 memB		
C306: 354 memB		
C306: 355 memB		
C306: 356 memB		
C306: 357 memB		
C306: 358 memB		
C306: 359 memB		
C306: 360 memB		
C306: 361 memB		
C306: 362 memB		
C306: 363 memB		
C306: 364 memB		
C306: 365 memB		
C306: 366 memB		
C306: 367 memB		
C306: 368 memB		
C306: 369 memB		
C306: 370 memB		
C306: 371 memB		
C306: 372 memB		
C306: 373 memB		
C306: 374 memB		
C306: 375 memB		
C306: 376 memB		
C306: 377 memB		
C306: 378 memB		
C306: 379 memB		
C306: 380 memB		
C306: 381 memB		
C306: 382 memB		
C306: 383 memB		
C306: 384 memB		
C306: 385 memB		
C306: 386 memB		
C306: 387 memB		
C306: 388 memB		
C306: 389 memB		
C306: 390 memB		
C306: 391 memB		
C306: 392 memB		
C306: 393 memB		
C306: 394 memB		
C306: 395 memB		
C306: 396 memB		
C306: 397 memB		
C306: 398 memB		
C306: 399 memB		
C306: 400 memB		
C306: 401 memB		
C306: 402 memB		
C306: 403 memB		
C306: 404 memB		
C306: 405 memB		
C306: 406 memB		
C306: 407 memB		
C306: 408 memB		
C306: 409 memB		
C306: 410 memB		
C306: 411 memB		
C306: 412 memB		
C306: 413 memB		
C306: 414 memB		
C306: 415 memB		
C306: 416 memB		
C306: 417 memB		
C306: 418 memB		
C306: 419 memB		
C306: 420 memB		
C306: 421 memB		
C306: 422 memB		
C306: 423 memB		
C306: 424 memB		
C306: 425 memB		
C306: 426 memB		
C306: 427 memB		
C306: 428 memB		
C306: 429 memB		
C306: 430 memB		
C306: 431 memB		
C306: 432 memB		
C306: 433 memB		
C306: 434 memB		
C306: 435 memB		
C306: 436 memB		
C306: 437 memB		
C306: 438 memB		
C306: 439 memB		
C306: 440 memB		
C306: 441 memB		
C306: 442 memB		
C306: 443 memB		
C306: 444 memB		
C306: 445 memB		
C306: 446 memB		
C306: 447 memB		
C306: 448 memB		
C306: 449 memB		
C306: 450 memB		
C306: 451 memB		
C306: 452 memB		
C306: 453 memB		
C306: 454 memB		
C306: 455 memB		
C306: 456 memB		
C306: 457 memB		
C306: 458 memB		

24 BANGER2	Apple //c diagnostics	26-JUL-85	PAGE 89	24 BANGER2	Apple //c diagnostics	26-JUL-85	PAGE 90
C472:38	98 MEMERROR	sec		C4C4:A2 82	143 BADSMTC	ldx #2	
C473:AA	99 BADBITS	tax		C4C5:7A	144	ply	
C474:AD 13 C8	100	ldx	r-dremd	C4C6:88	145	php	anticipate MMU error
C475:88	101	clv	main or aux mem?	C4C7:9D 6C C8	146	ldx	
C476:2C 2A C8	102	bit	branch if primary bank	C4C8:88	147	bsw1ch1	
C477:88	103	bs1ts1		C4C9:88	148	php	
C478:88	104	bs1ts1		C4D0:88	149	bcc	branch if not IOU error
C479:AA 88	105	ldy #6	try to clear video screen	C4D1:98 6F C8	150	ldx	anticipate IOU error
C480:99 FE BF	106	clrst	loadr-2,y	C4D2:8D 86 C8	151	cpv #6	compare with where we left off
C481:99 86 C8	107	sta	loadr-6,y	C4D3:88 88	152	bcc	skip if MMU
C482:88	108	sta		C4D4:88 88	153	bcc	
C483:88	109	dey		C4D5:88 84	154	bcc	skip if GLU (loudis or dhires failure)
C484:88	110	clrst		C4E0:88 11	155	cpv #811	
C485:88	111	bne		C4E1:98 03	156	bcc	skip if IOU
C486:88	112	tax		C4E2:8D 72 C8	157	bsw1ch3	IOU error (loudis failure)
C487:88	113	tax		C4E3:8D 88 #5	158	bsw1ch2	
C488:88	114	sta	txpage1	C4E4:88	159	ata	amess-6,x
C489:88	115	sta	txpage1	C4E5:88	160	bcc	amess-6,x
C490:88	116	sta	txpage1	C4E6:88	161	bcc	amess-6,x
C491:88	117	sta	txpage1	C4E7:88	162	bcc	amess-6,x
C492:88	118	sta	txpage1	C4E8:88	163	bcc	amess-6,x
C493:88	119	sta	txpage1	C4E9:88	164	bcc	amess-6,x
C494:88	120	sta	txpage1	C4EA:88	165	bcc	amess-6,x
C495:88	121	sta	txpage1	C4EB:88	166	bcc	amess-6,x
C496:88	122	sta	txpage1	C4EC:88	167	bcc	amess-6,x
C497:88	123	sta	txpage1	C4ED:88	168	bcc	amess-6,x
C498:88	124	sta	txpage1	C4EE:88	169	bcc	amess-6,x
C499:88	125	sta	txpage1	C4EF:88	170	bcc	amess-6,x
C500:88	126	sta	txpage1	C4F0:88	171	bcc	amess-6,x
C501:88	127	sta	txpage1	C4F1:88	172	bcc	amess-6,x
C502:88	128	sta	txpage1	C4F2:88	173	bcc	amess-6,x
C503:88	129	sta	txpage1	C4F3:88	174	bcc	amess-6,x
C504:88	130	sta	txpage1	C4F4:88	175	bcc	amess-6,x
C505:88	131	sta	txpage1	C4F5:88	176	bcc	amess-6,x
C506:88	132	sta	txpage1	C4F6:88	177	bcc	amess-6,x
C507:88	133	sta	txpage1	C4F7:88	178	bcc	amess-6,x
C508:88	134	sta	txpage1	C4F8:88	179	bcc	amess-6,x
C509:88	135	sta	txpage1	C4F9:88	180	bcc	amess-6,x
C510:88	136	sta	txpage1	C4FA:88	181	bcc	amess-6,x
C511:88	137	sta	txpage1	C4FB:88	182	bcc	amess-6,x
C512:88	138	sta	txpage1	C4FC:88	183	bcc	amess-6,x
C513:88	139	sta	txpage1	C4FD:88	184	bcc	amess-6,x
C514:88	140	sta	txpage1	C4FE:88	185	bcc	amess-6,x
C515:88	141	sta	txpage1	C4FF:88	186	bcc	amess-6,x
C516:88	142	sta	txpage1	C500:88	187	bcc	amess-6,x
C517:88	143	sta	txpage1	C501:88	188	bcc	amess-6,x
C518:88	144	sta	txpage1	C502:88	189	bcc	amess-6,x
C519:88	145	sta	txpage1	C503:88	190	bcc	amess-6,x
C520:88	146	sta	txpage1	C504:88	191	bcc	amess-6,x
C521:88	147	sta	txpage1	C505:88	192	bcc	amess-6,x
C522:88	148	sta	txpage1	C506:88	193	bcc	amess-6,x
C523:88	149	sta	txpage1	C507:88	194	bcc	amess-6,x
C524:88	150	sta	txpage1	C508:88	195	bcc	amess-6,x
C525:88	151	sta	txpage1	C509:88	196	bcc	amess-6,x
C526:88	152	sta	txpage1	C510:88	197	bcc	amess-6,x
C527:88	153	sta	txpage1	C511:88	198	bcc	amess-6,x
C528:88	154	sta	txpage1	C512:88	199	bcc	amess-6,x
C529:88	155	sta	txpage1	C513:88	200	bcc	amess-6,x
C530:88	156	sta	txpage1	C514:88	201	bcc	amess-6,x
C531:88	157	sta	txpage1	C515:88	202	bcc	amess-6,x
C532:88	158	sta	txpage1	C516:88	203	bcc	amess-6,x
C533:88	159	sta	txpage1	C517:88	204	bcc	amess-6,x
C534:88	160	sta	txpage1	C518:88	205	bcc	amess-6,x
C535:88	161	sta	txpage1	C519:88	206	bcc	amess-6,x
C536:88	162	sta	txpage1	C520:88	207	bcc	amess-6,x
C537:88	163	sta	txpage1	C521:88	208	bcc	amess-6,x
C538:88	164	sta	txpage1	C522:88	209	bcc	amess-6,x
C539:88	165	sta	txpage1	C523:88	210	bcc	amess-6,x
C540:88	166	sta	txpage1	C524:88	211	bcc	amess-6,x
C541:88	167	sta	txpage1	C525:88	212	bcc	amess-6,x
C542:88	168	sta	txpage1	C526:88	213	bcc	amess-6,x
C543:88	169	sta	txpage1	C527:88	214	bcc	amess-6,x
C544:88	170	sta	txpage1	C528:88	215	bcc	amess-6,x
C545:88	171	sta	txpage1	C529:88	216	bcc	amess-6,x
C546:88	172	sta	txpage1	C530:88	217	bcc	amess-6,x
C547:88	173	sta	txpage1	C531:88	218	bcc	amess-6,x
C548:88	174	sta	txpage1	C532:88	219	bcc	amess-6,x
C549:88	175	sta	txpage1	C533:88	220	bcc	amess-6,x
C550:88	176	sta	txpage1	C534:88	221	bcc	amess-6,x
C551:88	177	sta	txpage1	C535:88	222	bcc	amess-6,x
C552:88	178	sta	txpage1	C536:88	223	bcc	amess-6,x
C553:88	179	sta	txpage1	C537:88	224	bcc	amess-6,x
C554:88	180	sta	txpage1	C538:88	225	bcc	amess-6,x
C555:88	181	sta	txpage1	C539:88	226	bcc	amess-6,x
C556:88	182	sta	txpage1	C540:88	227	bcc	amess-6,x
C557:88	183	sta	txpage1	C541:88	228	bcc	amess-6,x
C558:88	184	sta	txpage1	C542:88	229	bcc	amess-6,x
C559:88	185	sta	txpage1	C543:88	230	bcc	amess-6,x
C560:88	186	sta	txpage1	C544:88	231	bcc	amess-6,x
C561:88	187	sta	txpage1	C545:88	232	bcc	amess-6,x
C562:88	188	sta	txpage1	C546:88	233	bcc	amess-6,x
C563:88	189	sta	txpage1	C547:88	234	bcc	amess-6,x
C564:88	190	sta	txpage1	C548:88	235	bcc	amess-6,x
C565:88	191	sta	txpage1	C549:88	236	bcc	amess-6,x
C566:88	192	sta	txpage1	C550:88	237	bcc	amess-6,x
C567:88	193	sta	txpage1	C551:88	238	bcc	amess-6,x
C568:88	194	sta	txpage1	C552:88	239	bcc	amess-6,x
C569:88	195	sta	txpage1	C553:88	240	bcc	amess-6,x
C570:88	196	sta	txpage1	C554:88	241	bcc	amess-6,x
C571:88	197	sta	txpage1	C555:88	242	bcc	amess-6,x
C572:88	198	sta	txpage1	C556:88	243	bcc	amess-6,x
C573:88	199	sta	txpage1	C557:88	244	bcc	amess-6,x
C574:88	200	sta	txpage1	C558:88	245	bcc	amess-6,x
C575:88	201	sta	txpage1	C559:88	246	bcc	amess-6,x
C576:88	202	sta	txpage1	C560:88	247	bcc	amess-6,x
C577:88	203	sta	txpage1	C561:88	248	bcc	amess-6,x
C578:88	204	sta	txpage1	C562:88	249	bcc	amess-6,x
C579:88	205	sta	txpage1	C563:88	250	bcc	amess-6,x
C580:88	206	sta	txpage1	C564:88	251	bcc	amess-6,x
C581:88	207	sta	txpage1	C565:88	252	bcc	amess-6,x
C582:88	208	sta	txpage1	C566:88	253	bcc	amess-6,x
C583:88	209	sta	txpage1	C567:88	254	bcc	amess-6,x
C584:88	210	sta	txpage1	C568:88	255	bcc	amess-6,x
C585:88	211	sta	txpage1	C569:88	256	bcc	amess-6,x
C586:88	212	sta	txpage1	C570:88	257	bcc	amess-6,x
C587:88	213	sta	txpage1	C571:88	258	bcc	amess-6,x
C588:88	214	sta	txpage1	C572:88	259	bcc	amess-6,x
C589:88	215	sta	txpage1	C573:88	260	bcc	amess-6,x
C590:88	216	sta	txpage1	C574:88	261	bcc	amess-6,x
C591:88	217	sta	txpage1	C575:88	262	bcc	amess-6,x
C592:88	218	sta	txpage1	C576:88	263	bcc	amess-6,x
C593:88	219	sta	txpage1	C577:88	264	bcc	amess-6,x
C594:88	220	sta	txpage1	C578:88	265	bcc	amess-6,x
C595:88	221	sta	txpage1	C579:88	266	bcc	amess-6,x
C596:88	222	sta	txpage1	C580:88	267	bcc	amess-6,x
C597:88	223	sta	txpage1	C581:88	268	bcc	amess-6,x
C598:88	224	sta	txpage1	C582:88	269	bcc	amess-6,x
C599:88	225	sta	txpage1	C583:88	270	bcc	amess-6,x
C600:88	226	sta	txpage1	C584:88	271	bcc	amess-6,x
C601:88	227	sta	txpage1	C585:88	272	bcc	amess-6,x
C602:88	228	sta	txpage1	C586:88	273	bcc	amess-6,x
C603:88	229	sta	txpage1	C587:88	274	bcc	amess-6,x
C604:88	230	sta	txpage1	C588:88	275	bcc	amess-6,x
C605:88	231	sta	txpage1	C589:88	276	bcc	amess-6,x
C606:88	232	sta	txpage1	C590:88	277	bcc	amess-6,x
C607:88	233	sta	txpage1	C591:88	278	bcc	amess-6,x
C608:88	234	sta	txpage1	C592:88	279	bcc	amess-6,x
C609:88	235	sta	txpage1	C593:88	280	bcc	amess-6,x
C610:88	236	sta	txpage1	C594:88	281	bcc	amess-6,x
C611:88	237	sta	txpage1	C595:88	282	bcc	amess-6,x
C612:88	238	sta	txpage1	C596:88	283	bcc	amess-6,x
C613:88	239	sta	txpage1	C597:88	284	bcc	amess-6,x
C614:88	240	sta	txpage1	C598:88	285	bcc	amess-6,x
C615:88	241	sta	txpage1	C599:88	286	bcc	amess-6,x
C616:88	242	sta	txpage1	C600:88	287	bcc	amess-6,x
C617:88	243	sta	txpage1	C601:88	288	bcc	amess-6,x
C618:88	244	sta	txpage1	C602:88	289	bcc	amess-6,x
C619:88	245	sta	txpage1	C603:88	290	bcc	amess-6,x
C620:88	246	sta	txpage1	C604:88	291	bcc	amess-6,x
C621:88	247	sta	txpage1	C605:88	292	bcc	amess-6,x
C622:88	248	sta	txpage1	C606:88	293	bcc	amess-6,x
C623:88	249	sta	txpage1	C607:88	294	bcc	amess-6,x
C624:88	250	sta	txpage1	C608:88	295	bcc	amess-6,x
C625:88	251	sta	txpage1	C609:88	296	bcc	amess-6,x
C626:88	252	sta	txpage1	C610:88	297	bcc	amess-6,x
C627:88	253	sta	txpage1	C611:88	298	bcc	amess-6,x
C628:88	254	sta	txpage1	C612:88	299	bcc	amess-6,x
C629:88	255	sta	txpage1	C613:88	300	bcc	amess-6,x
C630:88	256	sta	txpage1	C614:88	301	bcc	amess-6,x
C631:88	257	sta	txpage1	C615:88	302	bcc	amess-6,x
C632:88	258	sta	txpage1	C616:88	303	bcc	amess-6,x
C633:88	259	sta	txpage1	C617:88	304	bcc	amess-6,x
C634:88	260	sta	txpage1	C618:88	305	bcc	amess-6,x
C635:88	261	sta	txpage1	C619:88	306	bcc	amess-6,x
C636:88	262	sta	txpage1	C620:88	307	bcc	amess-6,x
C637:88	263	sta	txpage1	C6			

[illegible]

Command processor for serial & comm 26-JUL-85 PAGE 94

26 COMMAND

```

3 *****
3 * The command routine now supports 5 new 2-character commands.
4 *****
4 * commands enable or disable a feature of the serial port and are
5 * derived from their equivalent in the super serial card for the //.
6 *****
7 * The new commands are as follows:
8 * L - send LF out after CR
9 * X - detect XOFF, and wait for XON
10 * F - accept keyboard input
11 * A - ignore LF in either CR
12 * C - auto CR when column count > printer width
13 *****
13 * Usage of location $779 (port 1) and $77A (port 2) are as
14 * follows:
15 * bit 7 - echo output to screen if on
16 * bit 6 - generate LF after CR if on
17 * bit 5 - generate LF after XON if on
18 * bit 4 - accept XOFF if on
19 * bit 3 - accept LF in either CR if on
20 * bit 2 - a character was received through the ACIA and is in
21 * location $5FE (port 1) or $57E (port 2) if on
22 * bit 1 - XOFF is accepted, awaiting XON if on
23 * bit 0 - a character was received through the serial port if off
24 *****
25 char-OR equ $8D need an "upper case" space
26 ucspc equ $8B character
27 ucspc equ $8B
28 *****
29 command pha sermode,x
30 bit 7 sermode,x
31 bit 6 ldy sermode,x
32 bit 5 beq nocmd,x
33 bit 4 ser mode,x
34 bit 3 ser eschar,x
35 bit 2 asl A
36 bit 1 bne nocmd
37 bit 0 command1
38 bit 7 ldy nocmd
39 bit 6 ldy #cmdcr
40 bit 5 ldy cursor
41 bit 4 jmp cominit1
42 bit 3 sec
43 bit 2 pla
44 bit 1 nocmd2
45 bit 0 rts
46 *****
47 incmd equ devno2,x
48 bit 7 devno2,x
49 bit 6 and
50 bit 5 and $5F
51 *****
52 pha sermode,x
53 bit 7 lda sermode,x
54 bit 6 bne incmd2
55 bit 5 pla
56 bit 4 incmd1
57 bit 3 equ
58 bit 2 incmd2
59 bit 1 pla
60 bit 0 cmp
61 bit 7 incmd3
62 bit 6 bne incmd3
63 bit 5 cbc
64 bit 4 pla
65 bit 3 incmd2
66 bit 2 lda sermode,x
67 bit 1 lda pha
68 bit 0 and #7
69 bit 7 sta temp
70 bit 6 lda
71 bit 5 pla
72 bit 4 and $F8
73 bit 3 sta sermode,x
74 bit 2 lda
75 bit 1 and
76 bit 0 and
77 *****
78 *****
79 * Diagnostic routine tables
80 setv equ 93,67,43,41,7
81 bpl 80
82 swtbl8 dfb 989,983,985,989,991,$7F,$5F
83 dfb 989,983,985,989,991,$7F,$5F
84 swtbl1 dfb 989,983,985,989,991,$7F,$5F
85 dfb 989,983,985,989,991,$7F,$5F
86 swtbl1 dfb 989,983,985,989,991,$7F,$5F
87 dfb 989,983,985,989,991,$7F,$5F
88 swtbl1 dfb 989,983,985,989,991,$7F,$5F
89 dfb 989,983,985,989,991,$7F,$5F
90 swtbl1 dfb 989,983,985,989,991,$7F,$5F
91 dfb 989,983,985,989,991,$7F,$5F
92 swtbl1 dfb 989,983,985,989,991,$7F,$5F
93 dfb 989,983,985,989,991,$7F,$5F
94 swtbl1 dfb 989,983,985,989,991,$7F,$5F
95 dfb 989,983,985,989,991,$7F,$5F
96 swtbl1 dfb 989,983,985,989,991,$7F,$5F
97 dfb 989,983,985,989,991,$7F,$5F
98 swtbl1 dfb 989,983,985,989,991,$7F,$5F
99 dfb 989,983,985,989,991,$7F,$5F
100 *****
101 *****
102 *****
103 *****
104 *****
105 *****
106 *****
107 *****
108 *****
109 *****
110 *****
111 *****
112 *****
113 *****
114 *****
115 *****
116 *****
117 *****
118 *****
119 *****
120 *****
121 *****
122 *****
123 *****
124 *****
125 *****
126 *****
127 *****
128 *****
129 *****
130 *****
131 *****
132 *****
133 *****
134 *****
135 *****
136 *****
137 *****
138 *****
139 *****
140 *****
141 *****
142 *****
143 *****
144 *****
145 *****
146 *****
147 *****
148 *****
149 *****
150 *****
151 *****
152 *****
153 *****
154 *****
155 *****
156 *****
157 *****
158 *****
159 *****
160 *****
161 *****
162 *****
163 *****
164 *****
165 *****
166 *****
167 *****
168 *****
169 *****
170 *****
171 *****
172 *****
173 *****
174 *****
175 *****
176 *****
177 *****
178 *****
179 *****
180 *****
181 *****
182 *****
183 *****
184 *****
185 *****
186 *****
187 *****
188 *****
189 *****
190 *****
191 *****
192 *****
193 *****
194 *****
195 *****
196 *****
197 *****
198 *****
199 *****
200 *****
201 *****
202 *****
203 *****
204 *****
205 *****
206 *****
207 *****
208 *****
209 *****
210 *****
211 *****
212 *****
213 *****
214 *****
215 *****
216 *****
217 *****
218 *****
219 *****
220 *****
221 *****
222 *****
223 *****
224 *****
225 *****
226 *****
227 *****
228 *****
229 *****
230 *****
231 *****
232 *****
233 *****
234 *****
235 *****
236 *****
237 *****
238 *****
239 *****
240 *****
241 *****
242 *****
243 *****
244 *****
245 *****
246 *****
247 *****
248 *****
249 *****
250 *****
251 *****
252 *****
253 *****
254 *****
255 *****
256 *****
257 *****
258 *****
259 *****
260 *****
261 *****
262 *****
263 *****
264 *****
265 *****
266 *****
267 *****
268 *****
269 *****
270 *****
271 *****
272 *****
273 *****
274 *****
275 *****
276 *****
277 *****
278 *****
279 *****
280 *****
281 *****
282 *****
283 *****
284 *****
285 *****
286 *****
287 *****
288 *****
289 *****
290 *****
291 *****
292 *****
293 *****
294 *****
295 *****
296 *****
297 *****
298 *****
299 *****
300 *****
301 *****
302 *****
303 *****
304 *****
305 *****
306 *****
307 *****
308 *****
309 *****
310 *****
311 *****
312 *****
313 *****
314 *****
315 *****
316 *****
317 *****
318 *****
319 *****
320 *****
321 *****
322 *****
323 *****
324 *****
325 *****
326 *****
327 *****
328 *****
329 *****
330 *****
331 *****
332 *****
333 *****
334 *****
335 *****
336 *****
337 *****
338 *****
339 *****
340 *****
341 *****
342 *****
343 *****
344 *****
345 *****
346 *****
347 *****
348 *****
349 *****
350 *****
351 *****
352 *****
353 *****
354 *****
355 *****
356 *****
357 *****
358 *****
359 *****
360 *****
361 *****
362 *****
363 *****
364 *****
365 *****
366 *****
367 *****
368 *****
369 *****
370 *****
371 *****
372 *****
373 *****
374 *****
375 *****
376 *****
377 *****
378 *****
379 *****
380 *****
381 *****
382 *****
383 *****
384 *****
385 *****
386 *****
387 *****
388 *****
389 *****
390 *****
391 *****
392 *****
393 *****
394 *****
395 *****
396 *****
397 *****
398 *****
399 *****
400 *****
401 *****
402 *****
403 *****
404 *****
405 *****
406 *****
407 *****
408 *****
409 *****
410 *****
411 *****
412 *****
413 *****
414 *****
415 *****
416 *****
417 *****
418 *****
419 *****
420 *****
421 *****
422 *****
423 *****
424 *****
425 *****
426 *****
427 *****
428 *****
429 *****
430 *****
431 *****
432 *****
433 *****
434 *****
435 *****
436 *****
437 *****
438 *****
439 *****
440 *****
441 *****
442 *****
443 *****
444 *****
445 *****
446 *****
447 *****
448 *****
449 *****
450 *****
451 *****
452 *****
453 *****
454 *****
455 *****
456 *****
457 *****
458 *****
459 *****
460 *****
461 *****
462 *****
463 *****
464 *****
465 *****
466 *****
467 *****
468 *****
469 *****
470 *****
471 *****
472 *****
473 *****
474 *****
475 *****
476 *****
477 *****
478 *****
479 *****
480 *****
481 *****
482 *****
483 *****
484 *****
485 *****
486 *****
487 *****
488 *****
489 *****
490 *****
491 *****
492 *****
493 *****
494 *****
495 *****
496 *****
497 *****
498 *****
499 *****
500 *****
501 *****
502 *****
503 *****
504 *****
505 *****
506 *****
507 *****
508 *****
509 *****
510 *****
511 *****
512 *****
513 *****
514 *****
515 *****
516 *****
517 *****
518 *****
519 *****
520 *****
521 *****
522 *****
523 *****
524 *****
525 *****
526 *****
527 *****
528 *****
529 *****
530 *****
531 *****
532 *****
533 *****
534 *****
535 *****
536 *****
537 *****
538 *****
539 *****
540 *****
541 *****
542 *****
543 *****
544 *****
545 *****
546 *****
547 *****
548 *****
549 *****
550 *****
551 *****
552 *****
553 *****
554 *****
555 *****
556 *****
557 *****
558 *****
559 *****
560 *****
561 *****
562 *****
563 *****
564 *****
565 *****
566 *****
567 *****
568 *****
569 *****
570 *****
571 *****
572 *****
573 *****
574 *****
575 *****
576 *****
577 *****
578 *****
579 *****
580 *****
581 *****
582 *****
583 *****
584 *****
585 *****
586 *****
587 *****
588 *****
589 *****
590 *****
591 *****
592 *****
593 *****
594 *****
595 *****
596 *****
597 *****
598 *****
599 *****
600 *****
601 *****
602 *****
603 *****
604 *****
605 *****
606 *****
607 *****
608 *****
609 *****
610 *****
611 *****
612 *****
613 *****
614 *****
615 *****
616 *****
617 *****
618 *****
619 *****
620 *****
621 *****
622 *****
623 *****
624 *****
625 *****
626 *****
627 *****
628 *****
629 *****
630 *****
631 *****
632 *****
633 *****
634 *****
635 *****
636 *****
637 *****
638 *****
6
```

D44B:5B	72	plb	!get character back	D6C5:38	144	enable	equ	!got a 2-chr command &E
D44C:DA	74	lax	!above x Cn's on stack	D6C6:90	145	seq	!set carry	
D44D:AE F8 86	75	lax	!get index to command's 1st chr	D6C7:10	146	dfrb	!bcc to skip next byte (the CLC)	
D44E:1C9 45	76	cnr	!is it an E?	D6C7:10	147	disable	!got a 2-chr command &D	
D44F:1B 71	77	beq	!yes is it a D?	D6C7:10	148	clb	!clear carry	
D450:1B 71	78	beq	!yes is it a D?	D6C7:10	149	clb	!clear carry	
D451:1B 71	79	beq	!yes is it a D?	D6C7:10	150	clb	!clear carry	
D452:1B 71	80	plb	!retrieve X-Cn Cold X still hanging	D6C8:F0 27	D8F4	beq	!if X=0 then command is LE or LD	
D453:DA 36 86	81	phx	!push it back to keep stack neat	D6D0:E0 84	D112	beq	!so just make it act like L or K	
D454:1B 71	82	phx	!compare result of comparison factor bit	D6D0:F0 41	D112	beq	!if X=4 then command is CE or CD	
D455:AE F8 86	84	lax	!retrieve result of comparison factor bit	D6D1:	155	beq	!skip if so	
D456:1B 71	85	plb	!char to cmd char followed by nother	D6D1:	156	beq	!skip if so	
D457:1B 71	86	beq	!yes it's 1-chr cmd followed by nother	D6D1:8A	160	txe	!clear carry for arithmetic	
D458:1C9 4D	87	beq	!is it a (guess what) CR?	D6D2:10	161	clb	!multiply index by 2	
D459:1B 71	88	beq	!yes a 1-chr command	D6D4:69 83	D162	asl	!add 3 to get mask index	
D45A:2nu11 equ	89	beq	!come here for unimplemented but	D6D5:AA	164	tax	!put mask index in x	
D45B:FA 79 86	90	plb	!legal 2-chr cmd off stack	D6D6:50 81	D8DB	plb	!get carry backable so X is ready	
D45C:1B 71	91	plb	!pull x (Ch) off stack	D6D7:1E B8 83	167	inx	!cmd was Disable so inc X to next	
D45D:1B 71	92	plb	!restore non-cmd-mode cursor	D6D8:4C 39 D1	168	ready	!mask	
D45E:1E B8 83	93	asl	!clear cmd-mode bit (bit 7 of sermode)	D6DE:	D8DE	170	cdone	!sermode bit 8 tells whether to set
D45F:1E B8 83	94	lax	!by shifting out bit 7 & shifting in	D6DE:BD B8 83	171	lde	!so get it cmd mode	
D460:1B 71	95	bra	!turn marking character not handled	D6E1:4A	172	lde	!shift bit 8 to carry	
D461:1B 71	96	beq	!come here if get eschar after LXF	D6E2:1B D1	D8B5	bcs	!if set, start new cmd mode	
D462:FA 79 86	98	plb	!or T	D6E4:AD 79 86	174	lde	!restore the cursor	
D463:1B 71	99	plb	!need X-Cn to set bit 8 of sermode	D6E7:1B D1	D8B7	sta	!if fall through to csmet with carry	
D464:1B 71	100	plb	!bit 8 was 0, is now 1 - means new cmd mode	D6EA:80	176	cmst	!clear	
D465:1B 71	101	lax	!reload (again) X= index to cmd's	D6EB:1E B8 83	177	asl	!set command mode according to carry	
D466:1B 71	102	lax	!first chr if 2-chr cmd turns out 1	D6EC:28	178	plp	!leaves carry clear	
D467:1B 71	103	lax	!come here if 2-chr cmd turns out 1	D6EF:7E B8 83	179	ror	!character handled	
D468:1B 71	104	lax	!get command chr	D6F3:60	181	plb	!because carry clear...	
D469:1B 71	105	lax	!treat it as if we just got it	D6F4:	D8F4	183	cmd21	!come here to handle LE & LD
D470:1B 71	106	lax	!isn't command mode, not 2-chrs tho	D6F4:A9 4C	184	lde	!make LE look like L	
D471:1B 71	107	lax	!check 5 possible 2-chr cmds	D6F6:28	185	plp	!get P back with carry indicating E	
D472:1B 71	108	lax	!is it there?	D6F7:B0 96	D88F	186	bcs	!carry set means it was an E
D473:1B 71	109	lax	!yes, need to flag it for next time	D6F9:A3 48	187	lde	!make LD look like K	
D474:1B 71	110	lax	!nope	D6FB:80 92	D8BF	188	bra	!back to 1
D475:1B 71	111	lax	!try next if there is one	D6FD:8A	189	cmd2found	!txe	
D476:1B 71	112	lax	!check 13 commands	D6FE:1D B8 83	192	bra	!copy top 2 bits of sermode	
D477:1B 71	113	lax	!Right char?	D6FF:1D B8 83	193	bra	!set bit 3 - 2-chr-command-mode flag	
D478:1B 71	114	lax	!We didn't find it	D184:9D B8 83	194	sta	!sermode now holds index to 2-chr command issued	
D479:1B 71	115	lax	!if char is cntl char	D187:38	195	sec	!set carry so we stay in command	
D480:1B 71	116	lax	!it can be the new cmd char	D188:08	196	bra	!for next time	
D481:1B 71	117	lax	!branch if not cntl character	D18A:A9 D1	198	cmfnd	!get hi byte of where to go	
D482:1B 71	118	lax	!Save cmd char, drop thru cldig to	D18C:48	199	phx	!save it on stack	
D483:1B 71	119	lax	!zip it down to 8n if char was a	D18E:48	FS D1	201	phx	!get to byte of where to go
D484:1B 71	120	lax	!digit	D18F:68	202	plb	!go there by RTSing	
D485:1B 71	121	lax	!if not a digit, it is unexpected	D192:28	204	cmd-c	!create status to check carry bit	
D486:1B 71	122	lax	!intruder branch	D194:68	205	plb	!create slot number in x	
D487:1B 71	123	lax	!A = A + 18 * current number	D196:9E B8 84	D8E7	cmf c1	!create slot number in x	
D488:1B 71	124	lax	!C=8 on first entry	D198:80 C3	D8E8	plb	!CD is same as PWDTH=8, no CR	
D489:1B 71	125	lax	!not starting new cmd mode, just save	D19A:80 C3	D8E9	bra	!we're done here	
D490:1B 71	126	lax	!start new cmd mode here	D19B:80 C3	D8EA	lax	!get y index into aux screenholes	
D491:1B 71	127	lax	!get sermode	D19C:28	210	lax	!get more default PWDTH	
D492:1B 71	128	lax	!clear bits 8-5 (starting a new cmd	D19D:9D B8 84	212	lax	!we're done here	
D493:1B 71	129	lax	!clear bits 8-5 (starting a new cmd	D19E:9D B8 84	213	bra	!we're done here	
D494:1B 71	130	lax	!clear bits 8-5 (starting a new cmd	D19F:9D B8 84	214	bra	!we're done here	
D495:1B 71	131	lax	!clear bits 8-5 (starting a new cmd	D1A0:9D B8 84	215	bra	!we're done here	
D496:1B 71	132	lax	!clear bits 8-5 (starting a new cmd	D1A1:9D B8 84	216	bra	!we're done here	
D497:1B 71	133	lax	!clear bits 8-5 (starting a new cmd	D1A2:9D B8 84	217	bra	!we're done here	
D498:1B 71	134	lax	!clear bits 8-5 (starting a new cmd	D1A3:9D B8 84	218	bra	!we're done here	
D499:1B 71	135	lax	!clear bits 8-5 (starting a new cmd	D1A4:9D B8 84	219	bra	!we're done here	
D500:1B 71	136	lax	!clear bits 8-5 (starting a new cmd	D1A5:9D B8 84	220	bra	!we're done here	
D501:1B 71	137	lax	!clear bits 8-5 (starting a new cmd	D1A6:9D B8 84	221	bra	!we're done here	
D502:1B 71	138	lax	!clear bits 8-5 (starting a new cmd	D1A7:9D B8 84	222	bra	!we're done here	
D503:1B 71	139	lax	!clear bits 8-5 (starting a new cmd	D1A8:9D B8 84	223	bra	!we're done here	
D504:1B 71	140	lax	!clear bits 8-5 (starting a new cmd	D1A9:9D B8 84	224	bra	!we're done here	
D505:1B 71	141	lax	!clear bits 8-5 (starting a new cmd	D1AA:9D B8 84	225	bra	!we're done here	
D506:1B 71	142	lax	!clear bits 8-5 (starting a new cmd	D1AB:9D B8 84	226	bra	!we're done here	
D507:1B 71	143	lax	!clear bits 8-5 (starting a new cmd	D1AC:9D B8 84	227	bra	!we're done here	
D508:1B 71	144	lax	!clear bits 8-5 (starting a new cmd	D1AD:9D B8 84	228	bra	!we're done here	
D509:1B 71	145	lax	!clear bits 8-5 (starting a new cmd	D1AE:9D B8 84	229	bra	!we're done here	
D510:1B 71	146	lax	!clear bits 8-5 (starting a new cmd	D1AF:9D B8 84	230	bra	!we're done here	
D511:1B 71	147	lax	!clear bits 8-5 (starting a new cmd	D1B0:9D B8 84	231	bra	!we're done here	
D512:1B 71	148	lax	!clear bits 8-5 (starting a new cmd	D1B1:9D B8 84	232	bra	!we're done here	
D513:1B 71	149	lax	!clear bits 8-5 (starting a new cmd	D1B2:9D B8 84	233	bra	!we're done here	
D514:1B 71	150	lax	!clear bits 8-5 (starting a new cmd	D1B3:9D B8 84	234	bra	!we're done here	
D515:1B 71	151	lax	!clear bits 8-5 (starting a new cmd	D1B4:9D B8 84	235	bra	!we're done here	
D516:1B 71	152	lax	!clear bits 8-5 (starting a new cmd	D1B5:9D B8 84	236	bra	!we're done here	
D517:1B 71	153	lax	!clear bits 8-5 (starting a new cmd	D1B6:9D B8 84	237	bra	!we're done here	
D518:1B 71	154	lax	!clear bits 8-5 (starting a new cmd	D1B7:9D B8 84	238	bra	!we're done here	
D519:1B 71	155	lax	!clear bits 8-5 (starting a new cmd	D1B8:9D B8 84	239	bra	!we're done here	
D520:1B 71	156	lax	!clear bits 8-5 (starting a new cmd	D1B9:9D B8 84	240	bra	!we're done here	
D521:1B 71	157	lax	!clear bits 8-5 (starting a new cmd	D1BA:9D B8 84	241	bra	!we're done here	
D522:1B 71	158	lax	!clear bits 8-5 (starting a new cmd	D1BB:9D B8 84	242	bra	!we're done here	
D523:1B 71	159	lax	!clear bits 8-5 (starting a new cmd	D1BC:9D B8 84	243	bra	!we're done here	
D524:1B 71	160	lax	!clear bits 8-5 (starting a new cmd	D1BD:9D B8 84	244	bra	!we're done here	
D525:1B 71	161	lax	!clear bits 8-5 (starting a new cmd	D1BE:9D B8 84	245	bra	!we're done here	
D526:1B 71	162	lax	!clear bits 8-5 (starting a new cmd	D1BF:9D B8 84	246	bra	!we're done here	
D527:1B 71	163	lax	!clear bits 8-5 (starting a new cmd	D1C0:9D B8 84	247	bra	!we're done here	
D528:1B 71	164	lax	!clear bits 8-5 (starting a new cmd	D1C1:9D B8 84	248	bra	!we're done here	
D529:1B 71	165	lax	!clear bits 8-5 (starting a new cmd	D1C2:9D B8 84	249	bra	!we're done here	
D530:1B 71	166	lax	!clear bits 8-5 (starting a new cmd	D1C3:9D B8 84	250	bra	!we're done here	
D531:1B 71	167	lax	!clear bits 8-5 (starting a new cmd	D1C4:9D B8 84	251	bra	!we're done here	
D532:1B 71	168	lax	!clear bits 8-5 (starting a new cmd	D1C5:9D B8 84	252	bra	!we're done here	
D533:1B 71	169	lax	!clear bits 8-5 (starting a new cmd	D1C6:9D B8 84	253	bra	!we're done here	
D534:1B 71	170	lax	!clear bits 8-5 (starting a new cmd	D1C7:9D B8 84	254	bra	!we're done here	
D535:1B 71	171	lax	!clear bits 8-5 (starting a new cmd	D1C8:9D B8 84	255	bra	!we're done here	
D536:1B 71	172	lax	!clear bits 8-5 (starting a new cmd	D1C9:9D B8 84	256	bra	!we're done here	
D537:1B 71	173	lax	!clear bits 8-5 (starting a new cmd	D1CA:9D B8 84	257	bra	!we're done here	
D538:1B 71	174	lax	!clear bits 8-5 (starting a new cmd	D1CB:9D B8 84	258	bra	!we're done here	
D539:1B 71	175	lax	!clear bits 8-5 (starting a new cmd	D1CC:9D B8 84	259	bra	!we're done here	
D540:1B 71	176	lax	!clear bits 8-5 (starting a new cmd	D1CD:9D B8 84	260	bra	!we're done here	
D541:1B 71	177	lax	!clear bits 8-5 (starting a new cmd	D1CE:9D B8 84	261	bra	!we're done here	
D542:1B 71	178	lax	!clear bits 8-5 (starting a new cmd	D1CF:9D B8 84	262	bra	!we're done here	
D543:1B 71	179	lax	!clear bits 8-5 (starting a new cmd	D1D0:9D B8 84	263	bra	!we're done here	
D544:1B 71	180	lax	!clear bits 8-5 (starting a new cmd	D1D1:9D B8 84	264	bra	!we're done here	
D545:1B 71	181	lax	!clear bits 8-5 (starting a new cmd	D1D2:9D B8 84	265	bra	!we're done here	
D546:1B 71	182	lax	!clear bits 8-5 (starting a new cmd	D1D3:9D B8 84	266	bra	!we're done here	
D547:1B 71	183	lax	!clear bits 8-5 (starting a new cmd	D1D4:9D B8 84	267	bra	!we're done here	
D548:1B 71	184	lax	!clear bits 8-5 (starting a new cmd	D1D5:9D B8 84	268	bra	!we're done here	
D549:1B 71	185	lax	!clear bits 8-5 (starting a new cmd	D1D6:9D B8 84	269	bra	!we're done here	
D550:1B 71	186	lax	!clear bits 8-5 (starting a new cmd	D1D7:9D B8 84	270	bra	!we're done here	
D551:1B 71	187	lax	!clear bits 8-5 (starting a new cmd	D1D8:9D B8 84	271	bra	!we're done here	
D552:1B 71	188	lax	!clear bits 8-5 (starting a new cmd	D1D9:9D B8 84	272	bra	!we're done here	
D553:1B 71	189	lax	!clear bits 8-5 (starting a new cmd	D1DA:9D B8 84	273	bra	!we're done here	
D554:1B 71	190	lax	!clear bits 8-5 (starting a new cmd	D1DB:9D B8 84	274	bra	!we're done here	
D555:1B 71	191	lax	!clear bits 8-5 (starting a new cmd	D1DC:9D B8 84	275	bra	!we're done here	
D556:1B 71	192	lax	!clear bits 8-5 (starting a new cmd	D1DD:9D B8 84	276	bra	!we're done here	
D557:1B 71	193	lax	!clear bits 8-5 (starting a new cmd	D1DE:9D B8 84	277	bra	!we're done here	
D558:1B 71	194	lax	!clear bits 8-5 (starting a new cmd	D1DF:9D B8 84	278	bra	!we're done here	
D559:1B 71	195	lax	!clear bits 8-5 (starting a new cmd	D1E0:9D B8 84	279	bra	!we're done here	
D560:1B 71	196	lax	!clear bits 8-5 (starting a new cmd	D1E1:9D B8 84	280	bra	!we're done here	
D561:1B 71	197	lax	!clear bits 8-5 (starting a new cmd	D1E2:9D B8 84	281	bra	!we're done here	
D562:1B 71	198	lax	!clear bits 8-5 (starting a new cmd	D1E3:9D B8 84	282	bra	!we're done here	
D563:1B 71	199	lax	!clear bits 8-5 (starting a new cmd	D1E4:9D B8 84	283	bra	!we're done here	
D564:1B 71	200	lax	!clear bits 8-5 (starting a new cmd	D1E5:9D B8 84	284	bra	!we're done here	
D565:1B 71	201	lax	!clear bits 8-5 (starting a new cmd	D1E6:9D B8 84	285	bra	!we're done here	
D566:1B 71	202	lax	!clear bits 8-5 (starting a new cmd	D1E7:9D B8 84	286	bra	!we're done here	
D567:1B 71	203	lax	!clear bits 8-5 (starting a new cmd	D1E8:9D B8 84	287	bra	!we're done here	
D568:1B 71	204	lax	!clear bits 8-5 (starting a new cmd	D1E9:9D B8 84	288	bra	!we're done here	
D569:1B 71	205	lax	!clear bits 8-5 (starting a new cmd	D1EA:9D B8 84	289	bra	!we're done here	
D570:1B 71	206	lax	!clear bits 8-5 (starting a new cmd	D1EB:9D B8 84	290	bra	!we're done here	
D571:1B 71	207	lax	!clear bits 8-5 (starting a new cmd	D1EC:9D B8 84	291	bra	!we're done here	
D572:1B 71	208	lax	!clear bits 8-5 (starting a new cmd	D1ED:9D B8 84	292	bra	!we're done here	
D573:1B 71	209	lax	!clear bits 8-5 (starting a new cmd	D1EE:9D B8 84	293	bra	!we're done here	
D574:1B 71	210	lax	!clear bits 8-5 (starting a new cmd	D1EF:9D B8 84	294	bra	!we're done here	
D575:1B 71	211	lax	!clear bits 8-5 (starting a new cmd	D1F0:9D B8 84	295	bra	!we're done here	
D576:1B 71	212	lax	!clear bits 8-5 (starting a new cmd	D1F1:9D B8 84	296	bra	!we're done here	
D577:1B 71	213	lax	!clear bits 8-5 (starting a new cmd	D1F2:9D B8 84	297	bra	!we're done here	
D578:1B 71	214	lax	!clear bits 8-5 (starting a new cmd	D1F3:9D B8 84	298	bra	!we're done here	
D579:1B 71	215	lax	!clear bits 8-5 (starting a new cmd	D1F4:9D B				


```

26 COMMAND                                Command processor for serial & comm 26-JUL-85          PAGE 99
D20D:00 00 40 00                        351 mas2     dfb  00,000,040,000,000,000,000,000,000,000,000
D210:      D218                          353 cmdlist   equ  *
D218:49 4B 4C 4E                          354          asc  "IKLN"
D21C:0D      355                          355          dfb  00D
D220:42 44 50 51                          357          asc  "BDPQRSTZ"
D225:4C 58 46 4D                          358          asc  "LXPMC"
D225:4C 58 46 4D                          358          asc  ;2-chr commands' first chrs
D22A:      360 *****
D22A:      361 * R-GETALT is the same as GETALT in main rom. Only the
D22A:      362 * R-GETALT is the same as GETALT in main rom. Only the
D22A:      363 *****
D22A:AD 13 C0                          365 r-getalt  lda  r-dramrd
D22D:4A      366          esi
D22D:4A 10 C0                          367          lda  rd00col
D22E:4A 10 C0                          368          php
D230:0D 00 C0                          369          cld
D230:0D 00 C0                          370          stc  rdcardram
D235:0D 03 C0                          371          lda  0478-y
D238:09 78 04                          372          plp
D238:20      373          bcs  r-getalt1
D23C:00 03 D241                        374          r-dramrd
D240:00 03 D246                        375          r-getalt1
D241:00 03 D246                        376          r-getalt1
D243:0D 01 C0                          377          stc  rd00col
D246:00      378          rts
D247:03 07                          379 defidx2  dfb  3,7
D249:      55          include mdesic

```

```

27 MBASIC                                Mouse BASIC routines
D249:      01B7      2      ds  0D400-0,0

```



```

D450: *****
D451: 4 *****
D452: 5 * BASICIN - Input from basic
D453: 6 *****
D454: 7 * Creates -XXXX-YYYYV-SS
D455: 8 *****
D456: 9 * XXXX = X position
D457: 10 * YYYYV = Y position
D458: 11 * SS = Status
D459: 12 * = Key pressed
D460: 13 * 1 = Button pressed
D461: 14 * 2 = Button just released
D462: 15 * 3 = Button just released
D463: 16 * 4 = Button not pressed
D464: 17 *****
D465: 18 *****
D466: 19 basicin equ $hex11 y
D467: 20 idy $xw1 y
D468: 21 idy $xw1 y
D469: 22 sta $xw1
D470: 23 idy tbd
D471: 24 esi A
D472: 25 phi
D473: 26 sel
D474: 27 jsr xread2
D475: 28 idy #5
D476: 29 idy mouh
D477: 30 idy mouh
D478: 31 idy #12
D479: 32 idy mouh
D480: 33 idy mouh
D481: 34 idy mouh
D482: 35 jsr hexdec2
D483: 36 idy moustat
D484: 37 idy A
D485: 38 rol A
D486: 39 and #3
D487: 40 and #3
D488: 41 and #3
D489: 42 and #3
D490: 43 and #3
D491: 44 idy #16
D492: 45 jsr hexdec2
D493: 46 ply #17
D494: 47 idy #80
D495: 48 idy #80
D496: 49 putinbuf sta swr1s2
D497: 50 jmp swr1s2
D498: 51 *****
D499: 52 *****
D500: 53 *
D501: 54 * XREAD2 - duplicate of xread
D502: 55 *****
D503: 56 *****
D504: 57 equ $movarm
D505: 58 and $movarm
D506: 59 and $movarm
D507: 60 trb mouarm
D508: 61 mouarm
D509: 62 bml moubut
D510: 63 bml xrbut3
D511: 64 xrbut3
D512: 65 ora $80stat
D513: 66 bpl xrbut4
D514: 67 xrbut4
D515: 68 ora moustat
D516: 69 cpl
D517: 70 *****
D518: 71 *****
D519: 72 * HEXTODEC - Puts *0000 into the input buffer

```

```

D450: 73 * Inputs: A = Low byte of number
D451: 74 * X = High byte of number
D452: 75 * Y = Position of ones digit
D453: 76 *****
D454: 77 *****
D455: 78 hexdec2 equ #80
D456: 79 cpx #80
D457: 80 bcc hexdec2
D458: 81 bcc hexdec2
D459: 82 adc #80
D460: 83 adc #80
D461: 84 lxa
D462: 85 eor #80
D463: 86 edc #80
D464: 87 tax
D465: 88 pla
D466: 89 hexdec2
D467: 90 hexdec2
D468: 91 binh
D469: 92 idy #1
D470: 93 bcc hdp2
D471: 94 idy #1
D472: 95 hdp2
D473: 96 idy #1
D474: 97 sta inbuf+1 y
D475: 98 hdp2
D476: 99 idy #1
D477: 100 idy #1
D478: 101 idy #1
D479: 102 idy #1
D480: 103 idy #1
D481: 104 idy #1
D482: 105 idy #1
D483: 106 idy #1
D484: 107 idy #1
D485: 108 idy #1
D486: 109 idy #1
D487: 110 idy #1
D488: 111 idy #1
D489: 112 idy #1
D490: 113 idy #1
D491: 114 idy #1
D492: 115 idy #1
D493: 116 idy #1
D494: 117 idy #1
D495: 118 idy #1
D496: 119 idy #1
D497: 120 idy #1
D498: 121 idy #1
D499: 122 idy #1
D500: 123 idy #1
D501: 124 idy #1
D502: 125 idy #1
D503: 126 idy #1
D504: 127 idy #1
D505: 128 idy #1
D506: 129 idy #1
D507: 130 idy #1
D508: 131 idy #1
D509: 132 idy #1
D510: 133 idy #1
D511: 134 idy #1
D512: 135 idy #1
D513: 136 idy #1
D514: 137 idy #1
D515: 138 idy #1
D516: 139 idy #1
D517: 140 idy #1
D518: 141 idy #1
D519: 142 idy #1
D520: 143 idy #1
D521: 144 idy #1
D522: 145 idy #1
D523: 146 idy #1
D524: 147 idy #1
D525: 148 idy #1
D526: 149 idy #1
D527: 150 idy #1
D528: 151 idy #1
D529: 152 idy #1
D530: 153 idy #1
D531: 154 idy #1
D532: 155 idy #1
D533: 156 idy #1
D534: 157 idy #1
D535: 158 idy #1
D536: 159 idy #1
D537: 160 idy #1
D538: 161 idy #1
D539: 162 idy #1
D540: 163 idy #1
D541: 164 idy #1
D542: 165 idy #1
D543: 166 idy #1
D544: 167 idy #1
D545: 168 idy #1
D546: 169 idy #1
D547: 170 idy #1
D548: 171 idy #1
D549: 172 idy #1
D550: 173 idy #1
D551: 174 idy #1
D552: 175 idy #1
D553: 176 idy #1
D554: 177 idy #1
D555: 178 idy #1
D556: 179 idy #1
D557: 180 idy #1
D558: 181 idy #1
D559: 182 idy #1
D560: 183 idy #1
D561: 184 idy #1
D562: 185 idy #1
D563: 186 idy #1
D564: 187 idy #1
D565: 188 idy #1
D566: 189 idy #1
D567: 190 idy #1
D568: 191 idy #1
D569: 192 idy #1
D570: 193 idy #1
D571: 194 idy #1
D572: 195 idy #1
D573: 196 idy #1
D574: 197 idy #1
D575: 198 idy #1
D576: 199 idy #1
D577: 200 idy #1
D578: 201 idy #1
D579: 202 idy #1
D580: 203 idy #1
D581: 204 idy #1
D582: 205 idy #1
D583: 206 idy #1
D584: 207 idy #1
D585: 208 idy #1
D586: 209 idy #1
D587: 210 idy #1
D588: 211 idy #1
D589: 212 idy #1
D590: 213 idy #1
D591: 214 idy #1
D592: 215 idy #1
D593: 216 idy #1
D594: 217 idy #1
D595: 218 idy #1
D596: 219 idy #1
D597: 220 idy #1
D598: 221 idy #1
D599: 222 idy #1
D600: 223 idy #1
D601: 224 idy #1
D602: 225 idy #1
D603: 226 idy #1
D604: 227 idy #1
D605: 228 idy #1
D606: 229 idy #1
D607: 230 idy #1
D608: 231 idy #1
D609: 232 idy #1
D610: 233 idy #1
D611: 234 idy #1
D612: 235 idy #1
D613: 236 idy #1
D614: 237 idy #1
D615: 238 idy #1
D616: 239 idy #1
D617: 240 idy #1
D618: 241 idy #1
D619: 242 idy #1
D620: 243 idy #1
D621: 244 idy #1
D622: 245 idy #1
D623: 246 idy #1
D624: 247 idy #1
D625: 248 idy #1
D626: 249 idy #1
D627: 250 idy #1
D628: 251 idy #1
D629: 252 idy #1
D630: 253 idy #1
D631: 254 idy #1
D632: 255 idy #1
D633: 256 idy #1
D634: 257 idy #1
D635: 258 idy #1
D636: 259 idy #1
D637: 260 idy #1
D638: 261 idy #1
D639: 262 idy #1
D640: 263 idy #1
D641: 264 idy #1
D642: 265 idy #1
D643: 266 idy #1
D644: 267 idy #1
D645: 268 idy #1
D646: 269 idy #1
D647: 270 idy #1
D648: 271 idy #1
D649: 272 idy #1
D650: 273 idy #1
D651: 274 idy #1
D652: 275 idy #1
D653: 276 idy #1
D654: 277 idy #1
D655: 278 idy #1
D656: 279 idy #1
D657: 280 idy #1
D658: 281 idy #1
D659: 282 idy #1
D660: 283 idy #1
D661: 284 idy #1
D662: 285 idy #1
D663: 286 idy #1
D664: 287 idy #1
D665: 288 idy #1
D666: 289 idy #1
D667: 290 idy #1
D668: 291 idy #1
D669: 292 idy #1
D670: 293 idy #1
D671: 294 idy #1
D672: 295 idy #1
D673: 296 idy #1
D674: 297 idy #1
D675: 298 idy #1
D676: 299 idy #1
D677: 300 idy #1
D678: 301 idy #1
D679: 302 idy #1
D680: 303 idy #1
D681: 304 idy #1
D682: 305 idy #1
D683: 306 idy #1
D684: 307 idy #1
D685: 308 idy #1
D686: 309 idy #1
D687: 310 idy #1
D688: 311 idy #1
D689: 312 idy #1
D690: 313 idy #1
D691: 314 idy #1
D692: 315 idy #1
D693: 316 idy #1
D694: 317 idy #1
D695: 318 idy #1
D696: 319 idy #1
D697: 320 idy #1
D698: 321 idy #1
D699: 322 idy #1
D700: 323 idy #1
D701: 324 idy #1
D702: 325 idy #1
D703: 326 idy #1
D704: 327 idy #1
D705: 328 idy #1
D706: 329 idy #1
D707: 330 idy #1
D708: 331 idy #1
D709: 332 idy #1
D710: 333 idy #1
D711: 334 idy #1
D712: 335 idy #1
D713: 336 idy #1
D714: 337 idy #1
D715: 338 idy #1
D716: 339 idy #1
D717: 340 idy #1
D718: 341 idy #1
D719: 342 idy #1
D720: 343 idy #1
D721: 344 idy #1
D722: 345 idy #1
D723: 346 idy #1
D724: 347 idy #1
D725: 348 idy #1
D726: 349 idy #1
D727: 350 idy #1
D728: 351 idy #1
D729: 352 idy #1
D730: 353 idy #1
D731: 354 idy #1
D732: 355 idy #1
D733: 356 idy #1
D734: 357 idy #1
D735: 358 idy #1
D736: 359 idy #1
D737: 360 idy #1
D738: 361 idy #1
D739: 362 idy #1
D740: 363 idy #1
D741: 364 idy #1
D742: 365 idy #1
D743: 366 idy #1
D744: 367 idy #1
D745: 368 idy #1
D746: 369 idy #1
D747: 370 idy #1
D748: 371 idy #1
D749: 372 idy #1
D750: 373 idy #1
D751: 374 idy #1
D752: 375 idy #1
D753: 376 idy #1
D754: 377 idy #1
D755: 378 idy #1
D756: 379 idy #1
D757: 380 idy #1
D758: 381 idy #1
D759: 382 idy #1
D760: 383 idy #1
D761: 384 idy #1
D762: 385 idy #1
D763: 386 idy #1
D764: 387 idy #1
D765: 388 idy #1
D766: 389 idy #1
D767: 390 idy #1
D768: 391 idy #1
D769: 392 idy #1
D770: 393 idy #1
D771: 394 idy #1
D772: 395 idy #1
D773: 396 idy #1
D774: 397 idy #1
D775: 398 idy #1
D776: 399 idy #1
D777: 400 idy #1
D778: 401 idy #1
D779: 402 idy #1
D780: 403 idy #1
D781: 404 idy #1
D782: 405 idy #1
D783: 406 idy #1
D784: 407 idy #1
D785: 408 idy #1
D786: 409 idy #1
D787: 410 idy #1
D788: 411 idy #1
D789: 412 idy #1
D790: 413 idy #1
D791: 414 idy #1
D792: 415 idy #1
D793: 416 idy #1
D794: 417 idy #1
D795: 418 idy #1
D796: 419 idy #1
D797: 420 idy #1
D798: 421 idy #1
D799: 422 idy #1
D800: 423 idy #1
D801: 424 idy #1
D802: 425 idy #1
D803: 426 idy #1
D804: 427 idy #1
D805: 428 idy #1
D806: 429 idy #1
D807: 430 idy #1
D808: 431 idy #1
D809: 432 idy #1
D810: 433 idy #1
D811: 434 idy #1
D812: 435 idy #1
D813: 436 idy #1
D814: 437 idy #1
D815: 438 idy #1
D816: 439 idy #1
D817: 440 idy #1
D818: 441 idy #1
D819: 442 idy #1
D820: 443 idy #1
D821: 444 idy #1
D822: 445 idy #1
D823: 446 idy #1
D824: 447 idy #1
D825: 448 idy #1
D826: 449 idy #1
D827: 450 idy #1
D828: 451 idy #1
D829: 452 idy #1
D830: 453 idy #1
D831: 454 idy #1
D832: 455 idy #1
D833: 456 idy #1
D834: 457 idy #1
D835: 458 idy #1
D836: 459 idy #1
D837: 460 idy #1
D838: 461 idy #1
D839: 462 idy #1
D840: 463 idy #1
D841: 464 idy #1
D842: 465 idy #1
D843: 466 idy #1
D844: 467 idy #1
D845: 468 idy #1
D846: 469 idy #1
D847: 470 idy #1
D848: 471 idy #1
D849: 472 idy #1
D850: 473 idy #1
D851: 474 idy #1
D852: 475 idy #1
D853: 476 idy #1
D854: 477 idy #1
D855: 478 idy #1
D856: 479 idy #1
D857: 480 idy #1
D858: 481 idy #1
D859: 482 idy #1
D860: 483 idy #1
D861: 484 idy #1
D862: 485 idy #1
D863: 486 idy #1
D864: 487 idy #1
D865: 488 idy #1
D866: 489 idy #1
D867: 490 idy #1
D868: 491 idy #1
D869: 492 idy #1
D870: 493 idy #1
D871: 494 idy #1
D872: 495 idy #1
D873: 496 idy #1
D874: 497 idy #1
D875: 498 idy #1
D876: 499 idy #1
D877: 500 idy #1
D878: 501 idy #1
D879: 502 idy #1
D880: 503 idy #1
D881: 504 idy #1
D882: 505 idy #1
D883: 506 idy #1
D884: 507 idy #1
D885: 508 idy #1
D886: 509 idy #1
D887: 510 idy #1
D888: 511 idy #1
D889: 512 idy #1
D890: 513 idy #1
D891: 514 idy #1
D892: 515 idy #1
D893: 516 idy #1
D894: 517 idy #1
D895: 518 idy #1
D896: 519 idy #1
D897: 520 idy #1
D898: 521 idy #1
D899: 522 idy #1
D900: 523 idy #1
D901: 524 idy #1
D902: 525 idy #1
D903: 526 idy #1
D904: 527 idy #1
D905: 528 idy #1
D906: 529 idy #1
D907: 530 idy #1
D908: 531 idy #1
D909: 532 idy #1
D910: 533 idy #1
D911: 534 idy #1
D912: 535 idy #1
D913: 536 idy #1
D914: 537 idy #1
D915: 538 idy #1
D916: 539 idy #1
D917: 540 idy #1
D918: 541 idy #1
D919: 542 idy #1
D920: 543 idy #1
D921: 544 idy #1
D922: 545 idy #1
D923: 546 idy #1
D924: 547 idy #1
D925: 548 idy #1
D926: 549 idy #1
D927: 550 idy #1
D928: 551 idy #1
D929: 552 idy #1
D930: 553 idy #1
D931: 554 idy #1
D932: 555 idy #1
D933: 556 idy #1
D934: 557 idy #1
D935: 558 idy #1
D936: 559 idy #1
D937: 560 idy #1
D938: 561 idy #1
D939: 562 idy #1
D940: 563 idy #1
D941: 564 idy #1
D942: 565 idy #1
D943: 566 idy #1
D944: 567 idy #1
D945: 568 idy #1
D946: 569 idy #1
D947: 570 idy #1
D948: 571 idy #1
D949: 572 idy #1
D950: 573 idy #1
D951: 574 idy #1
D952: 575 idy #1
D953: 576 idy #1
D954: 577 idy #1
D955: 578 idy #1
D956: 579 idy #1
D957: 580 idy #1
D958: 581 idy #1
D959: 582 idy #1
D960: 583 idy #1
D961: 584 idy #1
D962: 585 idy #1
D963: 586 idy #1
D964: 587 idy #1
D965: 588 idy #1
D966: 589 idy #1
D967: 590 idy #1
D968: 591 idy #1
D969: 592 idy #1
D970: 593 idy #1
D971: 594 idy #1
D972: 595 idy #1
D973: 596 idy #1
D974: 597 idy #1
D975: 598 idy #1
D976: 599 idy #1
D977: 600 idy #1
D978: 601 idy #1
D979: 602 idy #1
D980: 603 idy #1
D981: 604 idy #1
D982: 605 idy #1
D983: 606 idy #1
D984: 607 idy #1
D985: 608 idy #1
D986: 609 idy #1
D987: 610 idy #1
D988: 611 idy #1
D989: 612 idy #1
D990: 613 idy #1
D991: 614 idy #1
D992: 615 idy #1
D993: 616 idy #1
D994: 617 idy #1
D995: 618 idy #1
D996: 619 idy #1
D997: 620 idy #1
D998: 621 idy #1
D999: 622 idy #1

```


28 BANNER	Apple //c Diagnostics	26-JUL-85	PAGE 183
D4A9:	3 * These routines test all 128K ram. All combinations of soft switches		
D4AB:	4 * applicable to the //c are tested and verified.		
D4AC:	5 * In the event of any failure, the diagnostic is halted. A message		
D4AD:	6 * is written to screen memory indicating the source of the failure.		
D4AE:	7 * When RAM fails the message is composed of "RAM ZP" (indicating detected in the first page of RAM) or "RAM" (meaning the other 63.75K).		
D4AF:	8 * followed by a binary representation of the failing bits set to "1".		
D4AG:	9 * For example, "RAM 0 1 1 0 0 0 0" indicates that bits 5 and 6 were detected as failing. To represent auxiliary memory, a "u" symbol is		
D4AH:	10 * printed preceding the message.		
D4AI:	11 * When the MMU or IOU fail, the message is simply "MMU" or "IOU".		
D4AJ:	12 * If the IOUDIS or DMRES switch fails, the message is "GLU".		
D4AK:	13 * The test will run continually for as long as the Open and Close		
D4AL:	14 * Apple keys remain depressed (or no keyboard is connected) and no failures are encountered. The message "System OK" will appear in		
D4AM:	15 * the middle of the screen when a successful cycle has been run and		
D4AN:	16 * either of the Apple keys are no longer depressed. Another cycle may be initiated by pressing both Apple keys again while this message is on the screen. To exit diagnostics, Control-Reset must be		
D4AO:	17 * pressed		
D4AP:	18 * without the Apple keys depressed.		
D4AQ:	19 * 20 * 21 * 22 * 23 * 24 * 25 * 26 * 27 * 28 * 29 * 30 * 31 * 32 * 33 * 34 * 35 * 36 * 37 * 38 * 39 * 40 * 41 * 42 * 43 * 44 * 45 * 46 * 47 * 48 * 49 * 50 * 51 * 52 * 53 * 54 * 55 * 56 * 57 * 58 * 59 * 60 * 61 * 62 * 63 * 64 * 65 * 66 * 67 * 68 * 69 * 70 * 71 * 72 * 73 * 74 * 75 * 76 * 77 * 78 * 79 * 80 * 81 * 82 * 83 * 84 * 85 * 86 * 87 * 88 * 89 * 90 * 91 * 92 * 93 * 94 * 95 * 96 * 97 * 98 * 99 *		
D4B2:	40 TSTZPG	ldy #64	
D4B3:	41 idx	#0	
D4B4:	42 cbc	atbl,y	
D4B5:	43 cbc	atbl,y	
D4B6:	44 stx	#00,x	
D4B7:	45 inx		
D4B8:	46 bne	zp1	
D4B9:	47 cbc	atbl,y	
D4BA:	48 adc	#00	
D4BB:	49 bne	ZERROR	
D4BC:	50 bne	zp2	
D4BD:	51 inx		
D4BE:	52 bne	zp2	
D4BF:	53 ror	a	
D4C0:	54 bit	rdvblbar	
D4C1:	55 bpl	zp3	
D4C2:	56 eor	#A5	
D4C3:	57 dey		
D4C4:	58 bpl	zp1	
D4C5:	59 bml	TSTMEN2	
D4C6:	60 eor	#00,x	
D4C7:	61 ZPERROR		
D4C8:	62 cbc	BADBITS	
D4C9:	63 jmp	TSTMEN	
D4CA:	64 TSTMEN2		
D4CB:	65 eor	#00,x	
D4CC:	66 znm		
D4CD:	67 jmp	swzsq12	
D4CE:	68 plv		
D4CF:	69 plv		
D4D0:	70 plv		
D4D1:	71 plv		
D4D2:	72 plv		
D4D3:	73 plv		
D4D4:	74 plv		
D4D5:	75 plv		
D4D6:	76 plv		
D4D7:	77 plv		
D4D8:	78 plv		
D4D9:	79 plv		
D4DA:	80 plv		
D4DB:	81 plv		
D4DC:	82 plv		
D4DD:	83 plv		
D4DE:	84 plv		
D4DF:	85 plv		
D4E0:	86 plv		
D4E1:	87 plv		
D4E2:	88 plv		
D4E3:	89 plv		
D4E4:	90 plv		
D4E5:	91 plv		
D4E6:	92 plv		
D4E7:	93 plv		
D4E8:	94 plv		
D4E9:	95 plv		
D4EA:	96 plv		
D4EB:	97 plv		
D4EC:	98 plv		
D4ED:	99 plv		
D4EE:	100 plv		
D4EF:	101 plv		
D4F0:	102 plv		
D4F1:	103 plv		
D4F2:	104 plv		
D4F3:	105 plv		
D4F4:	106 plv		
D4F5:	107 plv		
D4F6:	108 plv		
D4F7:	109 plv		
D4F8:	110 plv		
D4F9:	111 plv		
D4FA:	112 plv		
D4FB:	113 plv		
D4FC:	114 plv		
D4FD:	115 plv		
D4FE:	116 plv		
D4FF:	117 plv		
D500:	118 plv		
D501:	119 plv		
D502:	120 plv		
D503:	121 plv		
D504:	122 plv		
D505:	123 plv		
D506:	124 plv		
D507:	125 plv		
D508:	126 plv		
D509:	127 plv		
D50A:	128 plv		
D50B:	129 plv		
D50C:	130 plv		
D50D:	131 plv		
D50E:	132 plv		
D50F:	133 plv		
D510:	134 plv		
D511:	135 plv		
D512:	136 plv		
D513:	137 plv		
D514:	138 plv		
D515:	139 plv		
D516:	140 plv		
D517:	141 plv		
D518:	142 plv		
D519:	143 plv		
D51A:	144 plv		
D51B:	145 plv		
D51C:	146 plv		
D51D:	147 plv		
D51E:	148 plv		
D51F:	149 plv		
D520:	150 plv		
D521:	151 plv		
D522:	152 plv		
D523:	153 plv		
D524:	154 plv		
D525:	155 plv		
D526:	156 plv		
D527:	157 plv		
D528:	158 plv		
D529:	159 plv		
D52A:	160 plv		
D52B:	161 plv		
D52C:	162 plv		
D52D:	163 plv		
D52E:	164 plv		
D52F:	165 plv		
D530:	166 plv		
D531:	167 plv		
D532:	168 plv		
D533:	169 plv		
D534:	170 plv		
D535:	171 plv		
D536:	172 plv		
D537:	173 plv		
D538:	174 plv		
D539:	175 plv		
D53A:	176 plv		
D53B:	177 plv		
D53C:	178 plv		
D53D:	179 plv		
D53E:	180 plv		
D53F:	181 plv		
D540:	182 plv		
D541:	183 plv		
D542:	184 plv		
D543:	185 plv		
D544:	186 plv		
D545:	187 plv		
D546:	188 plv		
D547:	189 plv		
D548:	190 plv		
D549:	191 plv		
D54A:	192 plv		
D54B:	193 plv		
D54C:	194 plv		
D54D:	195 plv		
D54E:	196 plv		
D54F:	197 plv		
D550:	198 plv		
D551:	199 plv		
D552:	200 plv		
D553:	201 plv		
D554:	202 plv		
D555:	203 plv		
D556:	204 plv		
D557:	205 plv		
D558:	206 plv		
D559:	207 plv		
D55A:	208 plv		
D55B:	209 plv		
D55C:	210 plv		
D55D:	211 plv		
D55E:	212 plv		
D55F:	213 plv		
D560:	214 plv		
D561:	215 plv		
D562:	216 plv		
D563:	217 plv		
D564:	218 plv		
D565:	219 plv		
D566:	220 plv		
D567:	221 plv		
D568:	222 plv		
D569:	223 plv		
D56A:	224 plv		
D56B:	225 plv		
D56C:	226 plv		
D56D:	227 plv		
D56E:	228 plv		
D56F:	229 plv		
D570:	230 plv		
D571:	231 plv		
D572:	232 plv		
D573:	233 plv		
D574:	234 plv		
D575:	235 plv		
D576:	236 plv		
D577:	237 plv		
D578:	238 plv		
D579:	239 plv		
D57A:	240 plv		
D57B:	241 plv		
D57C:	242 plv		
D57D:	243 plv		
D57E:	244 plv		
D57F:	245 plv		
D580:	246 plv		
D581:	247 plv		
D582:	248 plv		
D583:	249 plv		
D584:	250 plv		
D585:	251 plv		
D586:	252 plv		
D587:	253 plv		
D588:	254 plv		
D589:	255 plv		
D58A:	256 plv		
D58B:	257 plv		
D58C:	258 plv		
D58D:	259 plv		
D58E:	260 plv		
D58F:	261 plv		
D590:	262 plv		
D591:	263 plv		
D592:	264 plv		
D593:	265 plv		
D594:	266 plv		
D595:	267 plv		
D596:	268 plv		
D597:	269 plv		
D598:	270 plv		
D599:	271 plv		
D59A:	272 plv		
D59B:	273 plv		
D59C:	274 plv		
D59D:	275 plv		
D59E:	276 plv		
D59F:	277 plv		
D5A0:	278 plv		
D5A1:	279 plv		
D5A2:	280 plv		
D5A3:	281 plv		
D5A4:	282 plv		
D5A5:	283 plv		
D5A6:	284 plv		
D5A7:	285 plv		
D5A8:	286 plv		
D5A9:	287 plv		
D5AA:	288 plv		
D5AB:	289 plv		
D5AC:	290 plv		
D5AD:	291 plv		
D5AE:	292 plv		
D5AF:	293 plv		
D5B0:	294 plv		
D5B1:	295 plv		
D5B2:	296 plv		
D5B3:	297 plv		
D5B4:	298 plv		
D5B5:	299 plv		
D5B6:	300 plv		
D5B7:	301 plv		
D5B8:	302 plv		
D5B9:	303 plv		
D5BA:	304 plv		
D5BB:	305 plv		
D5BC:	306 plv		
D5BD:	307 plv		
D5BE:	308 plv		
D5BF:	309 plv		
D5C0:	310 plv		
D5C1:	311 plv		
D5C2:	312 plv		
D5C3:	313 plv		
D5C4:	314 plv		
D5C5:	315 plv		
D5C6:	316 plv		
D5C7:	317 plv		
D5C8:	318 plv		
D5C9:	319 plv		
D5CA:	320 plv		
D5CB:	321 plv		
D5CC:	322 plv		
D5CD:	323 plv		
D5CE:	324 plv		
D5CF:	325 plv		
D5D0:	326 plv		
D5D1:	327 plv		
D5D2:	328 plv		
D5D3:	329 plv		
D5D4:	330 plv		
D5D5:	331 plv		
D5D6:	332 plv		
D5D7:	333 plv		
D5D8:	334 plv		
D5D9:	335 plv		
D5DA:	336 plv		
D5DB:	337 plv		
D5DC:	338 plv		
D5DD:	339 plv		
D5DE:	340 plv		
D5DF:	341 plv		
D5E0:	342 plv		
D5E1:	343 plv		
D5E2:	344 plv		
D5E3:	345 plv		
D5E4:	346 plv		
D5E5:	347 plv		
D5E6:	348 plv		
D5E7:	349 plv		
D5E8:	350 plv		
D5E9:	351 plv		
D5EA:	352 plv		
D5EB:	353 plv		
D5EC:	354 plv		
D5ED:	355 plv		
D5EE:	356 plv		
D5EF:	357 plv		
D5F0:	358 plv		
D5F1:	359 plv		
D5F2:	360 plv		
D5F3:	361 plv		
D5F4:	362 plv		
D5F5:	363 plv		
D5F6:	364 plv		
D5F7:	365 plv		
D5F8:	366 plv		
D5F9:	367 plv		
D5FA:	368 plv		
D5FB:	369 plv		
D5FC:	370 plv		
D5FD:	371 plv		
D5FE:	372 plv		
D5FF:	373 plv		
D600:	374 plv		
D601:	375 plv		
D602:	376 plv		
D603:	377 plv		
D604:	378 plv		
D605:	379 plv		
D606:	380 plv		
D607:	381 plv		
D608:	382 plv		
D609:	383 plv		
D60A:	384 plv		
D60B:	385 plv		
D60C:	386 plv		
D60D:	387 plv		
D60E:	388 plv		
D60F:	389 plv		
D610:	390 plv		

SORTED BY SYMBOL

29 SYMBOL TABLE26-JUL-85 PAGE 105

REPORTED BY SYMBOL

29 SYMPTOM TABLE

[illegible][illegible][illegible][illegible]

[illegible]

```

SOURCE FILE #01 ->CPCL
INCLUDE FILE #02 ->PC-EQUATES
INCLUDE FILE #03 ->PC-BOOTSPACE
INCLUDE FILE #04 ->PC-BOOT
INCLUDE FILE #05 ->PC-PACKET
INCLUDE FILE #06 ->PC-CREAD
INCLUDE FILE #07 ->PC-MAIN
0000:
0001: 0001 1 NOU
0002: 0001 2 IIC
0003: 0001 3 ROM
0004: 0000 4 TheOrg
0005: 1000 5 Version
0006: 0001 7 Ireq IIC
0007: 0001 9 else

```

```

01 CPCL
0000:
0001:
0002:
0003:
0004:
0005:
0006:
0007:
0008:
0009:
0010:
0011:
0012:
0013:
0014:
0015:
0016:
0017:
0018:
0019:
0020:
0021:
0022:
0023:
0024:
0025:
0026:
0027:
0028:
0029:
0030:
0031:
0032:

```

```

SmartPort code for A/c
11 fin
12 *
13 X6502
14
15: SSS M M AAAA RRRR TTTT PPPP 000 RRRR TTTT
16: MM MM A A R R T P P 0 0 R R R T
17: SSS MM MM AAAA RRRR T PPPP 0 0 RRRR T
18: S M M A A R R T P 0 0 R R R T
19: SSS M M A A R R T P 000 R R R T
20
21 * Protocol Converter Code for the Apple //c
22
23 *
24 * UniDisk 3.5 Driver Firmware Version 1.0
25 *
26 * Written by Michael Astina x6243 May 15, 1985
27 * Copyright Apple Computer, Inc. 1985
28 * All Rights Reserved
29 *
30 * MSB ON
31 *
32 *

```

```
34 *
35 include pc.equates
```

02 PC.EQUATES

EQUATES

```
0000: 000F 2 *
0001: 0000 3 PDIDByte equ $BF
0002: 0000 4 PCID2 equ $0
0003: 0000 5
0004: 0000 6
0005: 0000 7 *
0006: 0000 8 * Zero Page (tempa) *
0007: 0000 9 *
0008: 0000 10 *
0009: 0000 11 *
0010: 0000 12 dsect
0011: 0000 13 zeropage equ $0040
0012: 0000 14 *
0013: 0000 15 zeropage
0014: 0000 16 checksum dfb 0
0015: 0000 17 topbits dfb 0
0016: 0000 18 CMBCode dfb 0
0017: 0000 19 CMBCount equ *
0018: 0000 20 CMUnit dfb 0
0019: 0000 21 CMBuffer equ *
0020: 0000 22 CMDBuffer dfb 0
0021: 0000 23 CMDBufferh dfb 0
0022: 0000 24 CMDSCode equ *
0023: 0000 25 CMDBlock equ *
0024: 0000 26 CMDBlock1 dfb 0
0025: 0000 27 CMDBlock dfb 0
0026: 0000 28 CMDBlock2 dfb 0
0027: 0000 29 CMDSpare1 dfb 0
0028: 0000 30 CMDSpare2 dfb 0
0029: 0000 31 revbuf equ *
0030: 0000 32 gpcctr dfb 0
0031: 0000 33 CMDBlock equ *
0032: 0000 34 statbyte equ *
0033: 0000 35 bytecount equ *
0034: 0000 36 bytecount1 equ *
0035: 0000 37 next equ *
0036: 0000 38 next1 equ *
0037: 0000 39 BuType equ *
0038: 0000 40 bytecount equ *
0039: 0000 41 next2 dfb 0
0040: 0000 42 RPacketType equ *
0041: 0000 43 next3 dfb 0
0042: 0000 44 next4 equ *
0043: 0000 45 next1D dfb 0
0044: 0000 46 HostID equ *
0045: 0000 47 next5 dfb 0
0046: 0000 48 pointer equ *
0047: 0000 49 next6 dfb 0
0048: 0000 50 next7 dfb 0
0049: 0000 51 next8 dfb 0
0050: 0000 52 auxptr equ *
0051: 0000 53 buffer2 dw 0
0052: 0000 54 alot dfb 0
0053: 0000 55 temp equ *
0054: 0000 56 lbood dfb 0
0055: 0000 57 lbood1 dfb 0
0056: 0000 58 RPacketType dfb 0
0057: 0000 59 *
0058: 0000 60 *
0059: 0000 61 ZpSize equ *-zeropage
0060: 0000 62 *
0061: 0000 63 *
0062: 0000 64 *
0063: 0000 65 *
0064: 0000 66 ClearIDROMs equ $0FFF
0065: 0000 67 stack equ $100
0066: 0000 68 *
0067: 0000 69 *
0068: 0000 70 *
0069: 0000 71 *
0070: 0000 72 * Screenhole Storage *
```

;ProdOS attributes byte
;this means @ Liron card

;ProdOS parameter passing area

;Current target unit

```

0000: 73 * .....
0001: 74 .....
0002: 75 .....
0003: 76 * The screenhole layout is as follows:
0004: 77 .....
0005: 78 * //c
0006: 79 * .....
0007: 80 * ProfFlag $478+n
0008: 81 * Retry $4F8
0009: 82 * SHTemp1 $578+n
0010: 83 * SHTemp2 $578+n
0011: 84 * SHTemp3 $578+n
0012: 85 * Power1 $4F8+n
0013: 86 * Power2 $778+n
0014: 87 * NumDevices $7F8+n
0015: 88 * SvBCL $6F8
0016: 89 * SvBCH $778
0017: 90 *
0018: 91 * do ilc
0019: 92 * $1 sholes equ $473
0020: 93 * fin
0021: 94 *
0022: 95 *
0023: 96 *
0024: 97 ProfFlag equ sholes
0025: 98 Retry equ sholes+80
0026: 99 SHTemp1 equ sholes+100
0030: 100 SHTemp2 equ sholes+108
0031: 101 SHTemp3 equ sholes+109
0032: 102 SHTemp4 equ sholes+109
0033: 103 SHTemp5 equ sholes+109
0034: 104 SHTemp6 equ sholes+109
0035: 105 SHTemp7 equ sholes+109
0036: 106 SHTemp8 equ sholes+109
0037: 107 SHTemp9 equ sholes+109
0038: 108 SHTemp10 equ sholes+109
0039: 109 SHTemp11 equ sholes+109
0040: 110 SHTemp12 equ sholes+109
0041: 111 SHTemp13 equ sholes+109
0042: 112 SHTemp14 equ sholes+109
0043: 113 SHTemp15 equ sholes+109
0044: 114 SHTemp16 equ sholes+109
0045: 115 SHTemp17 equ sholes+109
0046: 116 SHTemp18 equ sholes+109
0047: 117 SHTemp19 equ sholes+109
0048: 118 SHTemp20 equ sholes+109
0049: 119 SHTemp21 equ sholes+109
0050: 120 SHTemp22 equ sholes+109
0051: 121 SHTemp23 equ sholes+109
0052: 122 SHTemp24 equ sholes+109
0053: 123 SHTemp25 equ sholes+109
0054: 124 SHTemp26 equ sholes+109
0055: 125 SHTemp27 equ sholes+109
0056: 126 SHTemp28 equ sholes+109
0057: 127 SHTemp29 equ sholes+109
0058: 128 SHTemp30 equ sholes+109
0059: 129 SHTemp31 equ sholes+109
0060: 130 SHTemp32 equ sholes+109
0061: 131 SHTemp33 equ sholes+109
0062: 132 SHTemp34 equ sholes+109
0063: 133 SHTemp35 equ sholes+109
0064: 134 SHTemp36 equ sholes+109
0065: 135 SHTemp37 equ sholes+109
0066: 136 SHTemp38 equ sholes+109
0067: 137 SHTemp39 equ sholes+109
0068: 138 SHTemp40 equ sholes+109
0069: 139 SHTemp41 equ sholes+109
0070: 140 SHTemp42 equ sholes+109
0071: 141 SHTemp43 equ sholes+109
0072: 142 SHTemp44 equ sholes+109
0073: 143 SHTemp45 equ sholes+109
0074: 144 SHTemp46 equ sholes+109
0075: 145 SHTemp47 equ sholes+109
0076: 146 SHTemp48 equ sholes+109
0077: 147 SHTemp49 equ sholes+109
0078: 148 SHTemp50 equ sholes+109
0079: 149 SHTemp51 equ sholes+109
0080: 150 SHTemp52 equ sholes+109
0081: 151 SHTemp53 equ sholes+109
0082: 152 SHTemp54 equ sholes+109
0083: 153 SHTemp55 equ sholes+109
0084: 154 SHTemp56 equ sholes+109
0085: 155 SHTemp57 equ sholes+109
0086: 156 SHTemp58 equ sholes+109
0087: 157 SHTemp59 equ sholes+109
0088: 158 SHTemp60 equ sholes+109
0089: 159 SHTemp61 equ sholes+109
0090: 160 SHTemp62 equ sholes+109
0091: 161 SHTemp63 equ sholes+109
0092: 162 SHTemp64 equ sholes+109
0093: 163 SHTemp65 equ sholes+109
0094: 164 SHTemp66 equ sholes+109
0095: 165 SHTemp67 equ sholes+109
0096: 166 SHTemp68 equ sholes+109
0097: 167 SHTemp69 equ sholes+109
0098: 168 SHTemp70 equ sholes+109
0099: 169 SHTemp71 equ sholes+109
0100: 170 SHTemp72 equ sholes+109
0101: 171 SHTemp73 equ sholes+109
0102: 172 SHTemp74 equ sholes+109
0103: 173 SHTemp75 equ sholes+109
0104: 174 SHTemp76 equ sholes+109
0105: 175 SHTemp77 equ sholes+109
0106: 176 SHTemp78 equ sholes+109
0107: 177 SHTemp79 equ sholes+109
0108: 178 SHTemp80 equ sholes+109
0109: 179 SHTemp81 equ sholes+109
0110: 180 SHTemp82 equ sholes+109
0111: 181 SHTemp83 equ sholes+109
0112: 182 SHTemp84 equ sholes+109
0113: 183 SHTemp85 equ sholes+109
0114: 184 SHTemp86 equ sholes+109
0115: 185 SHTemp87 equ sholes+109
0116: 186 SHTemp88 equ sholes+109
0117: 187 SHTemp89 equ sholes+109
0118: 188 SHTemp90 equ sholes+109
0119: 189 SHTemp91 equ sholes+109
0120: 190 SHTemp92 equ sholes+109
0121: 191 SHTemp93 equ sholes+109
0122: 192 SHTemp94 equ sholes+109
0123: 193 SHTemp95 equ sholes+109
0124: 194 SHTemp96 equ sholes+109
0125: 195 SHTemp97 equ sholes+109
0126: 196 SHTemp98 equ sholes+109
0127: 197 SHTemp99 equ sholes+109
0128: 198 SHTemp100 equ sholes+109
0129: 199 SHTemp101 equ sholes+109
0130: 200 SHTemp102 equ sholes+109
0131: 201 SHTemp103 equ sholes+109
0132: 202 SHTemp104 equ sholes+109
0133: 203 SHTemp105 equ sholes+109
0134: 204 SHTemp106 equ sholes+109
0135: 205 SHTemp107 equ sholes+109
0136: 206 SHTemp108 equ sholes+109
0137: 207 SHTemp109 equ sholes+109
0138: 208 SHTemp110 equ sholes+109
0139: 209 SHTemp111 equ sholes+109
0140: 210 SHTemp112 equ sholes+109
0141: 211 SHTemp113 equ sholes+109
0142: 212 SHTemp114 equ sholes+109
0143: 213 SHTemp115 equ sholes+109
0144: 214 SHTemp116 equ sholes+109
0145: 215 SHTemp117 equ sholes+109
0146: 216 SHTemp118 equ sholes+109
0147: 217 SHTemp119 equ sholes+109
0148: 218 SHTemp120 equ sholes+109
0149: 219 SHTemp121 equ sholes+109
0150: 220 SHTemp122 equ sholes+109
0151: 221 SHTemp123 equ sholes+109
0152: 222 SHTemp124 equ sholes+109
0153: 223 SHTemp125 equ sholes+109
0154: 224 SHTemp126 equ sholes+109
0155: 225 SHTemp127 equ sholes+109
0156: 226 SHTemp128 equ sholes+109
0157: 227 SHTemp129 equ sholes+109
0158: 228 SHTemp130 equ sholes+109
0159: 229 SHTemp131 equ sholes+109
0160: 230 SHTemp132 equ sholes+109
0161: 231 SHTemp133 equ sholes+109
0162: 232 SHTemp134 equ sholes+109
0163: 233 SHTemp135 equ sholes+109
0164: 234 SHTemp136 equ sholes+109
0165: 235 SHTemp137 equ sholes+109
0166: 236 SHTemp138 equ sholes+109
0167: 237 SHTemp139 equ sholes+109
0168: 238 SHTemp140 equ sholes+109
0169: 239 SHTemp141 equ sholes+109
0170: 240 SHTemp142 equ sholes+109
0171: 241 SHTemp143 equ sholes+109
0172: 242 SHTemp144 equ sholes+109
0173: 243 SHTemp145 equ sholes+109
0174: 244 SHTemp146 equ sholes+109
0175: 245 SHTemp147 equ sholes+109
0176: 246 SHTemp148 equ sholes+109
0177: 247 SHTemp149 equ sholes+109
0178: 248 SHTemp150 equ sholes+109
0179: 249 SHTemp151 equ sholes+109
0180: 250 SHTemp152 equ sholes+109
0181: 251 SHTemp153 equ sholes+109
0182: 252 SHTemp154 equ sholes+109
0183: 253 SHTemp155 equ sholes+109
0184: 254 SHTemp156 equ sholes+109
0185: 255 SHTemp157 equ sholes+109
0186: 256 SHTemp158 equ sholes+109
0187: 257 SHTemp159 equ sholes+109
0188: 258 SHTemp160 equ sholes+109
0189: 259 SHTemp161 equ sholes+109
0190: 260 SHTemp162 equ sholes+109
0191: 261 SHTemp163 equ sholes+109
0192: 262 SHTemp164 equ sholes+109
0193: 263 SHTemp165 equ sholes+109
0194: 264 SHTemp166 equ sholes+109
0195: 265 SHTemp167 equ sholes+109
0196: 266 SHTemp168 equ sholes+109
0197: 267 SHTemp169 equ sholes+109
0198: 268 SHTemp170 equ sholes+109
0199: 269 SHTemp171 equ sholes+109
0200: 270 SHTemp172 equ sholes+109
0201: 271 SHTemp173 equ sholes+109
0202: 272 SHTemp174 equ sholes+109
0203: 
```

Equates

148	packend	equ	\$C8
149	packend	equ	\$C8
150	packend	equ	\$C8
151	statemk	equ	\$B1
152	statemk	equ	\$B1
153	statemk	equ	\$B1
154	statemk	equ	\$B1
155	statemk	equ	\$B1
156	statemk	equ	\$B1
157	statemk	equ	\$B1
158	statemk	equ	\$B1
159	statemk	equ	\$B1
160	statemk	equ	\$B1
161	statemk	equ	\$B1
162	statemk	equ	\$B1
163	statemk	equ	\$B1
164	statemk	equ	\$B1
165	statemk	equ	\$B1
166	statemk	equ	\$B1
167	statemk	equ	\$B1
168	statemk	equ	\$B1
169	statemk	equ	\$B1
170	statemk	equ	\$B1
171	statemk	equ	\$B1
172	statemk	equ	\$B1
173	statemk	equ	\$B1
174	statemk	equ	\$B1
175	statemk	equ	\$B1
176	statemk	equ	\$B1
177	statemk	equ	\$B1
178	statemk	equ	\$B1
179	statemk	equ	\$B1
180	statemk	equ	\$B1
181	statemk	equ	\$B1
182	statemk	equ	\$B1
183	statemk	equ	\$B1
184	statemk	equ	\$B1
185	statemk	equ	\$B1
186	statemk	equ	\$B1
187	statemk	equ	\$B1
188	statemk	equ	\$B1
189	statemk	equ	\$B1
190	statemk	equ	\$B1
191	statemk	equ	\$B1
192	statemk	equ	\$B1
193	statemk	equ	\$B1
194	statemk	equ	\$B1
195	statemk	equ	\$B1
196	statemk	equ	\$B1
197	statemk	equ	\$B1
198	statemk	equ	\$B1
199	statemk	equ	\$B1
200	statemk	equ	\$B1
201	statemk	equ	\$B1
202	statemk	equ	\$B1
203	statemk	equ	\$B1
204	statemk	equ	\$B1
205	statemk	equ	\$B1
206	statemk	equ	\$B1
207	statemk	equ	\$B1
208	statemk	equ	\$B1
209	statemk	equ	\$B1
210	statemk	equ	\$B1
211	statemk	equ	\$B1
212	statemk	equ	\$B1
213	statemk	equ	\$B1
214	statemk	equ	\$B1
215	statemk	equ	\$B1
216	statemk	equ	\$B1
217	statemk	equ	\$B1
218	statemk	equ	\$B1
219	statemk	equ	\$B1
220	statemk	equ	\$B1
221	statemk	equ	\$B1
222	statemk	equ	\$B1
223	statemk	equ	\$B1
224	statemk	equ	\$B1
225	statemk	equ	\$B1
226	statemk	equ	\$B1
227	statemk	equ	\$B1
228	statemk	equ	\$B1
229	statemk	equ	\$B1
230	statemk	equ	\$B1
231	statemk	equ	\$B1
232	statemk	equ	\$B1
233	statemk	equ	\$B1
234	statemk	equ	\$B1
235	statemk	equ	\$B1
236	statemk	equ	\$B1
237	statemk	equ	\$B1
238	statemk	equ	\$B1
239	statemk	equ	\$B1
240	statemk	equ	\$B1
241	statemk	equ	\$B1
242	statemk	equ	\$B1
243	statemk	equ	\$B1
244	statemk	equ	\$B1
245	statemk	equ	\$B1
246	statemk	equ	\$B1
247	statemk	equ	\$B1
248	statemk	equ	\$B1
249	statemk	equ	\$B1
250	statemk	equ	\$B1
251	statemk	equ	\$B1
252	statemk	equ	\$B1
253	statemk	equ	\$B1
254	statemk	equ	\$B1
255	statemk	equ	\$B1
256	statemk	equ	\$B1
257	statemk	equ	\$B1
258	statemk	equ	\$B1
259	statemk	equ	\$B1
260	statemk	equ	\$B1
261	statemk	equ	\$B1
262	statemk	equ	\$B1
263	statemk	equ	\$B1
264	statemk	equ	\$B1
265	statemk	equ	\$B1
266	statemk	equ	\$B1
267	statemk	equ	\$B1
268	stat		

```

End of packet mark
Command Packet Identifier
Status Packet Identifier
Data Packet Identifier
No timer, asynch, latch
Get Device Specific Status
Get Dev Ctrl Block (modebits)
Return Newline Status
Get Device Info Block

```

10: The soft error bit in statbyte

```

C500: 947      ist on
C500: 948      *
C500: 949      * Here beginneth that code which resideth in the boot space
C500: 950      * at the time the card resteth in slot the fifth.
C500: 951      *
C500: 952      C$Borg equ *
C500: 953      *
C500: 954      * Auto Boot signature bytes
C500: 955      * This is also the boot (auto & PR$5) entry point.
C500: 956      *
C500: 957      ldx #20
C500: 958      ldx #40
C500: 959      ldx #80
C500: 960      *
C500: 961      cmp #0
C500: 962      do ilc*ROM
C500: 963      bcs BootC
C500: 964      else
C500: 965      fin
C500: 966      *
C500: 967      * Here is the ProDOS normal entry point
C500: 968      *
C500: 969      *
C500: 970      ProDSEntry equ *
C500: 971      *
C500: 972      * Set up so that ProFLAG will have the top bit set
C500: 973      *
C500: 974      sec
C500: 975      bcs *+3
C500: 976      *
C500: 977      * This is the MUXface entry point
C500: 978      *
C500: 979      MLEntry equ *
C500: 980      *
C500: 981      ldx #405
C500: 982      ror ProFLAG,x
C500: 983      cbc
C500: 984      *
C500: 985      * Now save mailot and clear all C$B00 ROMs
C500: 986      *
C500: 987      bootcases equ *
C500: 988      ldx #C5
C500: 989      ldx #800
C500: 990      ldx #80
C500: 991      ldx ClearIDROMs
C500: 992      *
C500: 993      do ilc*ROM
C500: 994      jmp SWPROT
C500: 995      ldx #405
C500: 996      ldx #0
C500: 997      else
C500: 998      fin
C500: 999      *
C500: 1000     *
C500: 1001     *
C500: 1002     *
C500: 1003     *
C500: 1004     *
C500: 1005     *
C500: 1006     *
C500: 1007     *
C500: 1008     *
C500: 1009     *
C500: 1010     *
C500: 1011     *
C500: 1012     *
C500: 1013     *
C500: 1014     *
C500: 1015     *
C500: 1016     *
C500: 1017     *
C500: 1018     *
C500: 1019     *
C500: 1020     *
C500: 1021     *
C500: 1022     *
C500: 1023     *
C500: 1024     *
C500: 1025     *
C500: 1026     *
C500: 1027     *
C500: 1028     *
C500: 1029     *
C500: 1030     *
C500: 1031     *
C500: 1032     *
C500: 1033     *
C500: 1034     *
C500: 1035     *
C500: 1036     *
C500: 1037     *
C500: 1038     *
C500: 1039     *
C500: 1040     *
C500: 1041     *
C500: 1042     *
C500: 1043     *
C500: 1044     *
C500: 1045     *
C500: 1046     *
C500: 1047     *
C500: 1048     *
C500: 1049     *
C500: 1050     *
C500: 1051     *
C500: 1052     *
C500: 1053     *
C500: 1054     *
C500: 1055     *
C500: 1056     *
C500: 1057     *
C500: 1058     *
C500: 1059     *
C500: 1060     *
C500: 1061     *
C500: 1062     *
C500: 1063     *
C500: 1064     *
C500: 1065     *
C500: 1066     *
C500: 1067     *
C500: 1068     *
C500: 1069     *
C500: 1070     *
C500: 1071     *
C500: 1072     *
C500: 1073     *
C500: 1074     *
C500: 1075     *
C500: 1076     *
C500: 1077     *
C500: 1078     *
C500: 1079     *
C500: 1080     *
C500: 1081     *
C500: 1082     *
C500: 1083     *
C500: 1084     *
C500: 1085     *
C500: 1086     *
C500: 1087     *
C500: 1088     *
C500: 1089     *
C500: 1090     *
C500: 1091     *
C500: 1092     *
C500: 1093     *
C500: 1094     *
C500: 1095     *
C500: 1096     *
C500: 1097     *
C500: 1098     *
C500: 1099     *
C500: 1100     *
C500: 1101     *
C500: 1102     *
C500: 1103     *
C500: 1104     *
C500: 1105     *
C500: 1106     *
C500: 1107     *
C500: 1108     *
C500: 1109     *
C500: 1110     *
C500: 1111     *
C500: 1112     *
C500: 1113     *
C500: 1114     *
C500: 1115     *
C500: 1116     *
C500: 1117     *
C500: 1118     *
C500: 1119     *
C500: 1120     *
C500: 1121     *
C500: 1122     *
C500: 1123     *
C500: 1124     *
C500: 1125     *
C500: 1126     *
C500: 1127     *
C500: 1128     *
C500: 1129     *
C500: 1130     *
C500: 1131     *
C500: 1132     *
C500: 1133     *
C500: 1134     *
C500: 1135     *
C500: 1136     *
C500: 1137     *
C500: 1138     *
C500: 1139     *
C500: 1140     *
C500: 1141     *
C500: 1142     *
C500: 1143     *
C500: 1144     *
C500: 1145     *
C500: 1146     *
C500: 1147     *
C500: 1148     *
C500: 1149     *
C500: 1150     *
C500: 1151     *
C500: 1152     *
C500: 1153     *
C500: 1154     *
C500: 1155     *
C500: 1156     *
C500: 1157     *
C500: 1158     *
C500: 1159     *
C500: 1160     *
C500: 1161     *
C500: 1162     *
C500: 1163     *
C500: 1164     *
C500: 1165     *
C500: 1166     *
C500: 1167     *
C500: 1168     *
C500: 1169     *
C500: 1170     *
C500: 1171     *
C500: 1172     *
C500: 1173     *
C500: 1174     *
C500: 1175     *
C500: 1176     *
C500: 1177     *
C500: 1178     *
C500: 1179     *
C500: 1180     *
C500: 1181     *
C500: 1182     *
C500: 1183     *
C500: 1184     *
C500: 1185     *
C500: 1186     *
C500: 1187     *
C500: 1188     *
C500: 1189     *
C500: 1190     *
C500: 1191     *
C500: 1192     *
C500: 1193     *
C500: 1194     *
C500: 1195     *
C500: 1196     *
C500: 1197     *
C500: 1198     *
C500: 1199     *
C500: 1200     *
C500: 1201     *
C500: 1202     *
C500: 1203     *
C500: 1204     *
C500: 1205     *
C500: 1206     *
C500: 1207     *
C500: 1208     *
C500: 1209     *
C500: 1210     *
C500: 1211     *
C500: 1212     *
C500: 1213     *
C500: 1214     *
C500: 1215     *
C500: 1216     *
C500: 1217     *
C500: 1218     *
C500: 1219     *
C500: 1220     *
C500: 1221     *
C500: 1222     *
C500: 1223     *
C500: 1224     *
C500: 1225     *
C500: 1226     *
C500: 1227     *
C500: 1228     *
C500: 1229     *
C500: 1230     *
C500: 1231     *
C500: 1232     *
C500: 1233     *
C500: 1234     *
C500: 1235     *
C500: 1236     *
C500: 1237     *
C500: 1238     *
C500: 1239     *
C500: 1240     *
C500: 1241     *
C500: 1242     *
C500: 1243     *
C500: 1244     *
C500: 1245     *
C500: 1246     *
C500: 1247     *
C500: 1248     *
C500: 1249     *
C500: 1250     *
C500: 1251     *
C500:
```



```

C523:      2 *
C523:      3 Bootcode equ *
C523:      4 slot
C523:      5 *
C525:      6 do ilc*ROM
C525:      7 idc *
C525:      8 M0C3
C525:      9 M0C3
C526:      10 M0C3
C526:      11 M0C3
C526:      12 M0C3
C526:      13 M0C3
C526:      14 M0C3
C526:      15 M0C3
C526:      16 M0C3
C526:      17 M0C3
C526:      18 M0C3
C526:      19 M0C3
C526:      20 M0C3
C526:      21 M0C3
C526:      22 M0C3
C526:      23 M0C3
C526:      24 M0C3
C526:      25 M0C3
C526:      26 M0C3
C526:      27 M0C3
C526:      28 M0C3
C526:      29 M0C3
C526:      30 M0C3
C526:      31 M0C3
C526:      32 M0C3
C526:      33 M0C3
C526:      34 M0C3
C526:      35 M0C3
C526:      36 M0C3
C526:      37 M0C3
C526:      38 M0C3
C526:      39 M0C3
C526:      40 M0C3
C526:      41 M0C3
C526:      42 M0C3
C526:      43 M0C3
C526:      44 M0C3
C526:      45 M0C3
C526:      46 M0C3
C526:      47 M0C3
C526:      48 M0C3
C526:      49 M0C3
C526:      50 M0C3
C526:      51 M0C3
C526:      52 M0C3
C526:      53 M0C3
C526:      54 M0C3
C526:      55 M0C3
C526:      56 M0C3
C526:      57 M0C3
C526:      58 M0C3
C526:      59 M0C3
C526:      60 M0C3
C526:      61 M0C3
C526:      62 M0C3
C526:      63 M0C3
C526:      64 M0C3
C526:      65 M0C3
C526:      66 M0C3
C526:      67 M0C3
C526:      68 M0C3
C526:      69 M0C3
C526:      70 M0C3
C526:      71 M0C3
C526:      72 M0C3
C526:      73 M0C3
C526:      74 M0C3
C526:      75 M0C3
C526:      76 M0C3
C526:      77 M0C3
C526:      78 M0C3
C526:      79 M0C3
C526:      80 M0C3
C526:      81 M0C3
C526:      82 M0C3
C526:      83 M0C3
C526:      84 M0C3
C526:      85 M0C3
C526:      86 M0C3
C526:      87 M0C3
C526:      88 M0C3
C526:      89 M0C3
C526:      90 M0C3
C526:      91 M0C3
C526:      92 M0C3
C526:      93 M0C3
C526:      94 M0C3
C526:      95 M0C3
C526:      96 M0C3
C526:      97 M0C3
C526:      98 M0C3
C526:      99 M0C3
C526:      100 M0C3

```

```

C576:      126 * This routine is called from the //c reset code. It forces a
C576:      127 * reset of the PC Bus.
C576:      128 *
C576:      129 do ilc*ROM
C576:      130 idc *
C576:      131 M0C3
C576:      132 M0C3
C576:      133 M0C3
C576:      134 M0C3
C576:      135 M0C3
C576:      136 M0C3
C576:      137 M0C3
C576:      138 M0C3
C576:      139 M0C3
C576:      140 M0C3
C576:      141 M0C3
C576:      142 M0C3
C576:      143 M0C3
C576:      144 M0C3
C576:      145 M0C3
C576:      146 M0C3
C576:      147 M0C3
C576:      148 M0C3
C576:      149 M0C3
C576:      150 M0C3
C576:      151 M0C3
C576:      152 M0C3
C576:      153 M0C3
C576:      154 M0C3
C576:      155 M0C3
C576:      156 M0C3
C576:      157 M0C3
C576:      158 M0C3
C576:      159 M0C3
C576:      160 M0C3
C576:      161 M0C3
C576:      162 M0C3
C576:      163 M0C3
C576:      164 M0C3
C576:      165 M0C3
C576:      166 M0C3
C576:      167 M0C3
C576:      168 M0C3
C576:      169 M0C3
C576:      170 M0C3
C576:      171 M0C3
C576:      172 M0C3
C576:      173 M0C3
C576:      174 M0C3
C576:      175 M0C3
C576:      176 M0C3
C576:      177 M0C3
C576:      178 M0C3
C576:      179 M0C3
C576:      180 M0C3
C576:      181 M0C3
C576:      182 M0C3
C576:      183 M0C3
C576:      184 M0C3
C576:      185 M0C3
C576:      186 M0C3
C576:      187 M0C3
C576:      188 M0C3
C576:      189 M0C3
C576:      190 M0C3
C576:      191 M0C3
C576:      192 M0C3
C576:      193 M0C3
C576:      194 M0C3
C576:      195 M0C3
C576:      196 M0C3
C576:      197 M0C3
C576:      198 M0C3
C576:      199 M0C3
C576:      200 M0C3

```


[illegible]

```

C980: 146 * Send first byte
C981: 147 *
C982: 148 *
C983: 149 *
C984: 150 *
C985: 151 *
C986: 152 *
C987: 153 *
C988: 154 *
C989: 155 *
C990: 156 *
C991: 157 *
C992: 158 *
C993: 159 *
C994: 160 *
C995: 161 *
C996: 162 *
C997: 163 *
C998: 164 *
C999: 165 *
C1000: 166 *
C1001: 167 *
C1002: 168 *
C1003: 169 *
C1004: 170 *
C1005: 171 *
C1006: 172 *
C1007: 173 *
C1008: 174 *
C1009: 175 *
C1010: 176 *
C1011: 177 *
C1012: 178 *
C1013: 179 *
C1014: 180 *
C1015: 181 *
C1016: 182 *
C1017: 183 *
C1018: 184 *
C1019: 185 *
C1020: 186 *
C1021: 187 *
C1022: 188 *
C1023: 189 *
C1024: 190 *
C1025: 191 *
C1026: 192 *
C1027: 193 *
C1028: 194 *
C1029: 195 *
C1030: 196 *
C1031: 197 *
C1032: 198 *
C1033: 199 *
C1034: 200 *
C1035: 201 *
C1036: 202 *
C1037: 203 *
C1038: 204 *
C1039: 205 *
C1040: 206 *
C1041: 207 *
C1042: 208 *
C1043: 209 *
C1044: 210 *
C1045: 211 *
C1046: 212 *
C1047: 213 *
C1048: 214 *
C1049: 215 *
C1050: 216 *
C1051: 217 *
C1052: 218 *
C1053: 219 *
C1054: 220 *
C1055: 221 *
C1056: 222 *
C1057: 223 *
C1058: 224 *
C1059: 225 *
C1060: 226 *
C1061: 227 *
C1062: 228 *
C1063: 229 *
C1064: 230 *
C1065: 231 *
C1066: 232 *
C1067: 233 *
C1068: 234 *
C1069: 235 *
C1070: 236 *
C1071: 237 *
C1072: 238 *
C1073: 239 *
C1074: 240 *
C1075: 241 *
C1076: 242 *
C1077: 243 *
C1078: 244 *
C1079: 245 *
C1080: 246 *
C1081: 247 *
C1082: 248 *
C1083: 249 *
C1084: 250 *
C1085: 251 *
C1086: 252 *
C1087: 253 *
C1088: 254 *
C1089: 255 *
C1090: 256 *
C1091: 257 *
C1092: 258 *
C1093: 259 *
C1094: 260 *
C1095: 261 *
C1096: 262 *
C1097: 263 *
C1098: 264 *
C1099: 265 *
C1100: 266 *
C1101: 267 *
C1102: 268 *
C1103: 269 *
C1104: 270 *
C1105: 271 *
C1106: 272 *
C1107: 273 *
C1108: 274 *
C1109: 275 *
C1110: 276 *
C1111: 277 *
C1112: 278 *
C1113: 279 *
C1114: 280 *
C1115: 281 *
C1116: 282 *
C1117: 283 *
C1118: 284 *
C1119: 285 *
C1120: 286 *
C1121: 287 *

```

85 PC PACKET	Send a CBus Packet	29-JUL-85 15:19 PAGE 15	85 PC PACKET	Send a CBus Packet	29-JUL-85 15:19 PAGE 16
C982:BD 8C C8	288 sd7 lda l6c1r,x		C9DC:	335 *	
C985:29 48 C982	289 and #848		C9DC:	336 *	* These routines are for wasting specific amounts of time
C987:DD F9 C982	290 bne sd7	:Still writing data	C9DC:	337 *	* This code segment should not cross page boundaries.
C988:9D 8D C8	291 ste l6set,x	:Back to sense mode (dummy write)	C9DC:20 E1 C9	338 waste32	jar waste14
C98C:	294 *	* Now wait until the drive acknowledges receipt of the	C9DF:EA	340 waste18	nop
C98C:	295 *	* string or until timeout	C9E1:EA	341 waste16	nop
C98C:AA 8A	297 ldy #byte02	:Load timeout to see bsy low	C9E1:EA	342 waste14	nop
C98E:88 C8	298 patch1 dey	:A little closer to an error	C9E3:00	344 waste12	rts
C98F:DD 88 C9C9	299 bne sd9	:There's still time	C9E3:	345 *	
C9C1:	300 *	* Too much time has elapsed. Drive didn't get string.	C9E3:4C C3 C9	346 markerr	equ *
C9C1:	302 *	* Report error in comm error byte		jmp	dberror
C9C1:A9 81	303 lda #noanswer				
C9C3:20 9A CA	304 dberror equ *				
C9C6:38	305 jar SetXN8	:For dberror entry			
C9C9:86 C9CF	306 sec	:Signal a problem			
C9C9:	308 bcs sd10				
C9C9:	309 *	* See if drive has acknowledged the bytes yet			
C9C9:	310 *				
C9C9:BD 8E C8	311 sd9 lda l7c1r,x	:Wait 'till /BSY lo			
C9CC:30 F8 C9BE	312 bmi patch1				
C9E1:	313 *				
C9E1:	315 *	* Finish the sequence			
C9E1:	316 *				
C9E1:18	317 cbc	:This is a normal exit			
C9E1:BD 88 C8	317 sd10 lda reqc1r,x	:Set REQ lo			
C9E2:BD 8C C8	318 lda l6c1r,x	:Back into read mode			
C9E5:	320 *				
C9E5:	320 *	* Pull back the bytecount in all cases			
C9E5:	321 *				
C9E5:68	322 rts				
C9E6:	323 *				
C9E6:	324 *				
C9E6:	324 *	* This table, when sent in reverse order, provides a			
C9E6:	325 *	* sync pattern used to synchronize the data stream with			
C9E6:	327 *	* the data stream. The first byte (last sent) is the			
C9E6:	328 *	* packet begin mark.			
C9E6:	329 *				
C9E6:	329 *				
C9E7:C3	331 preamble dfb	packetbeg			
C9E7:CF FC F3 CF	331 synctab dfb	0FF,0FC,0F3,\$CF,\$3F			
C9DC:	332 *				
C9DC:	333 *				

```

Receive a CBUS Packet      29-JUL-85   15:19 PAGE 11

*****
349 *****
350 .....
351 .....
352 .....
353 .....
354 REQ -----|2-----5|-----
355 .....
356 /BSV ---|1---3--4|----
357 .....
358 1) Drive signals ready to send packet
359 2) Data signal ready to receive data
360 3) Packet transmitted (sync, manr, IDs, date,
361    checksum [mg=1])
362 4) Drive signals packet dispatched
363 5) Host acknowledges receipt of packet
364 .....
365 The bytes are sent in slow mode (32 cycles/byte)
366 and the timing is critical. Branches which should
367 not cross page boundaries are marked.
368 .....
369 Input: buffer c- address where packet guts left
370 .....
371 Output: carry set- handshake error
372          clr- bytes received
373          A - error if carry set
374 .....
375 *****
376 grabstatus equ .....
377 ReceivePack equ .....
378 .....
379 Init the checksum .....
380 * .....
381 idc #000 .....
382 sta checksum .....
383 .....
384 Copy over buffer -> buffer2 .....
385 .....
386 idc buffer .....
387 sta buffer2 .....
388 lda buffer+1 .....
389 sta buffer+1 .....
390 .....
391 Set up the indirect pointer for jump to 2nd part of code
392 .....
393 .....
394 ifeq lhc#ROM ;Don't do in //c version
395 fnc .....
396 .....
397 jar enablechain ;Set X register to $N$B
398 .....
399 lda l6set,x ;Prep for sense mode
400 .....
401 Now wait for BSV to go hi, signalling 'ready w/ status'
402 .....
403 rdhi .....
404 bpl rdn1 ;Read sense
405 bpl rdn1 ;Wait til a high
406 .....
407 Signal Liron we're ready to receive
408 .....
409 lda request,x ;Raise /REQ
410 .....
411 Wait for a byte from Liron or timeout
412 .....
413 ldw @stalmco ;Max bytes 'til stat mart
414 .....
415 rdh2 .....
416 bpl rdn2 .....
417 .....
418 No Page Cross ***
419 .....
420 bnf mterr ;Didn't find a packet in time
421 .....
422 Is it the beginning of the packet?
423 .....

```

[illegible]

```

;Recycle handshake and set carry
;Carry set still

```


85 PC PACKET	Divide by 7 routine	29-JUL-85 15:19 PAGE 23	85 PC PACKET	Checksum Prepass	29-JUL-85 15:19 PAGE 24	
CB96:65 4C	adc	iGet new MOD value	CB98:	728 *	*****	
CB98:C9 97	cmp	!Is it too big?	CB98:	729 *	*****	
CB9A:98 02	bit	!> NO leave MOD - 0-3C	CB98:	730 *	PreCheck	Does the checksumming prepass
CB9C:E9 07	sbc	!Bring MOD under 7 - C still set	CB98:	731 *	Input:	Bytes in buffer to send
CB9E:65 4C	equ		CB98:	732 *	auxptr	Pointers to send process
CB9F:BD 55 CB	ldi	iGet DIV for this 2^n	CB98:	733 *	checksum	Pointers to send process
CB9A:65 4B	adc	!Add to DIV along with correction	CB98:	734 *	checksum	Pointers to send process
	grp7ctr	(C)	CB98:	735 *	checksum	Pointers to send process
CB9A:05 4B	sta	!Update the DIV	CB98:	736 *	*****	*****
CB97:CA	equ	!One less bit to deal with	CB98:	737 *	PreCheck equ *	*****
CB98:30 06	bni	!Escape after 6 times through loop	CB98:	738 *	PreCheck equ *	*****
CB9A:D0 E2	bne	!Take brnch 1st 5 loops	CB98:	739 *	Checksum any full pages	*****
CB9C:			CB98:	740 *	Checksum any full pages	*****
CB9C:98	tye	iGet back the last three bits	CB98:	741 *	Checksum any full pages	*****
CB9D:4C 95 CB	jmp	!Sixth pass add in remains	CB98:	742 *	Checksum any full pages	*****
CB9E:	equ *		CB98:	743 *	Checksum any full pages	*****
CB9F:			CB98:	744 *	Checksum any full pages	*****
CB9F:			CB98:	745 *	Checksum any full pages	*****
CB9F:			CB98:	746 *	Checksum any full pages	*****
CB9F:			CB98:	747 *	Checksum any full pages	*****
CB9F:			CB98:	748 *	Checksum any full pages	*****
CB9F:			CB98:	749 *	Checksum any full pages	*****
CB9F:			CB98:	750 *	Checksum any full pages	*****
CB9F:			CB98:	751 *	Checksum any full pages	*****
CB9F:			CB98:	752 *	Checksum any full pages	*****
CB9F:			CB98:	753 *	Checksum any full pages	*****
CB9F:			CB98:	754 *	Checksum any full pages	*****
CB9F:			CB98:	755 *	Checksum any full pages	*****
CB9F:			CB98:	756 *	Checksum any full pages	*****
CB9F:			CB98:	757 *	Checksum any full pages	*****
CB9F:			CB98:	758 *	Checksum any full pages	*****
CB9F:			CB98:	759 *	Checksum any full pages	*****
CB9F:			CB98:	760 *	Checksum any full pages	*****
CB9F:			CB98:	761 *	Checksum any full pages	*****
CB9F:			CB98:	762 *	Checksum any full pages	*****
CB9F:			CB98:	763 *	Checksum any full pages	*****
CB9F:			CB98:	764 *	Checksum any full pages	*****
CB9F:			CB98:	765 *	Checksum any full pages	*****
CB9F:			CB98:	766 *	Checksum any full pages	*****
CB9F:			CB98:	767 *	Checksum any full pages	*****
CB9F:			CB98:	768 *	Checksum any full pages	*****
CB9F:			CB98:	769 *	Checksum any full pages	*****
CB9F:			CB98:	770 *	Checksum any full pages	*****
CB9F:			CB98:	771 *	Checksum any full pages	*****
CB9F:			CB98:	772 *	Checksum any full pages	*****
CB9F:			CB98:	773 *	Checksum any full pages	*****
CB9F:			CB98:	774 *	Checksum any full pages	*****
CB9F:			CB98:	775 *	Checksum any full pages	*****
CB9F:			CB98:	776 *	Checksum any full pages	*****
CB9F:			CB98:	777 *	Checksum any full pages	*****
CB9F:			CB98:	778 *	Checksum any full pages	*****
CB9F:			CB98:	779 *	Checksum any full pages	*****
CB9F:			CB98:	780 *	Checksum any full pages	*****
CB9F:			CB98:	781 *	Checksum any full pages	*****
CB9F:			CB98:	782 *	Checksum any full pages	*****
CB9F:			CB98:	783 *	Checksum any full pages	*****
CB9F:			CB98:	784 *	Checksum any full pages	*****

Get topbits byte for odds

```

CBE1: .....
CBE1: 786 .....
CBE1: 787 .....
CBE1: 788 ..... Get topbits for odd bytes
CBE1: 789 ..... DetTopBits
CBE1: 790 ..... Also sets buffer2 pointer to pointer at groups of
CBE1: 791 ..... seven bytes.
CBE1: 792 .....
CBE1: 793 ..... Input: oddbytes ← # of "odd" bytes
CBE1: 794 ..... tbodd ← pointer to data
CBE1: 795 ..... tbodd ← topbits for odd bytes
CBE1: 796 ..... Output: buffer2 ← buffer+oddbytes
CBE1: 797 .....
CBE1: 798 .....
CBE1: 799 .....
CBE1: 800 .....
CBE1: 801 ..... DetTopBits equ *
CBE1: 802 .....
CBE1: 803 ..... ldy oddbytes
CBE1: 804 ..... dey #8
CBE1: 805 ..... lda #8
CBE1: 806 ..... sta tbodd
CBE1: 807 ..... lda (buffer).y
CBE1: 808 ..... esi a
CBE1: 809 ..... ror tbodd
CBE1: 810 ..... dey #1
CBE1: 811 ..... sec
CBE1: 812 ..... ror tbodd
CBE1: 813 .....
CBE1: 814 .....
CBE1: 815 ..... lda oddbytes
CBE1: 816 ..... cbc
CBE1: 817 ..... buffer
CBE1: 818 ..... sta buffer2
CBE1: 819 ..... lda buffer+1
CBE1: 820 ..... edc #8
CBE1: 821 ..... sta buffer2+1
CBE1: 822 .....
CBE1: 823 .....

```

Prime write pump

```

CC08: .....
CC08: 825 .....
CC08: 826 ..... Set up next buffer and topbits
CC08: 827 ..... Sun
CC08: 828 .....
CC08: 829 ..... Primes the pipe for the group of seven bytes routine
CC08: 830 ..... setting the topbits byte and the "next" buffer.
CC08: 831 ..... The routine also advances the buffer pointer by 7 to
CC08: 832 ..... prepare for the groups of seven transfer.
CC08: 833 .....
CC08: 834 ..... Input: buffer2 ← points to groups of 7 data
CC08: 835 ..... Output: next1,7 ← first 7 bytes in buffer
CC08: 836 ..... topbits ← MSBs of first 7 bytes
CC08: 837 .....
CC08: 838 .....
CC08: 839 .....
CC08: 840 Sun equ *
CC08: 841 .....
CC08: 842 ..... Copy first seven bytes into the pipeline
CC08: 843 ..... ldy #6
CC08: 844 ..... sec
CC08: 845 sun2
CC08: 846 lda (buffer2).y
CC08: 847 sta next1.y
CC08: 848 bml sun1
CC08: 849 cbc
CC08: 850 dev topbits
CC08: 851 bpl sun2
CC08: 852 ror
CC08: 853 sec
CC08: 854 ror topbits
CC08: 855
CC08: 856 ..... Advance the pointer
CC08: 857 .....
CC08: 858 ..... lda buffer2
CC08: 859 cbc
CC08: 860 adc #7
CC08: 861 sta buffer2
CC08: 862 bml sun3
CC08: 863 inc buffer2+1
CC08: 864 sun3
CC08: 865 equ *
CC08: 866 .....
CC08: 867 .....

```



```

CC1F: 869 *
CC1F: 870 * X is slot*16, Y is the desired mode
CC1F: 871 *
CC1F: 872 * Set up the IMM mode register. Extreme care should be taken
CC1F: 873 * before setting the mode byte, in indexed stores causes
CC1F: 874 * a bus error. Set before the timer, causes the IMM Rev A
CC1F: 875 * to
CC1F: 876 * pop the motor on, inhibiting the setting of the mode until the
CC1F: 877 * motor times out! We avoid this by setting the mode byte only
CC1F: 878 * if it's not what we want, and if it's not we stay here until we
CC1F: 879 * see that it is what we want.
CC1F: 880 *
CC1F: 881 SetIMMMode equ *
CC1F: 882 ldr monclr,x
CC1F: 883 ldr imm,x
CC1F: 884 ldr imm,x
CC1F: 885 bix careful
CC1F: 886 ldr imm,x
CC1F: 887 ldr imm,x
CC1F: 888 ldr imm,x
CC1F: 889 ldr imm,x
CC1F: 890 ldr imm,x
CC1F: 891 ldr imm,x
CC1F: 892 ldr imm,x
CC1F: 893 ldr imm,x
CC1F: 894 ldr imm,x
CC1F: 895 ldr imm,x
CC1F: 896 ldr imm,x
CC1F: 897 ldr imm,x
CC1F: 898 ldr imm,x
CC1F: 899 ldr imm,x
CC1F: 900 ldr imm,x
CC1F: 901 ldr imm,x
CC1F: 902 ldr imm,x
CC1F: 903 ldr imm,x
CC1F: 904 ldr imm,x
CC1F: 905 ldr imm,x
CC1F: 906 ldr imm,x
CC1F: 907 ldr imm,x
CC1F: 908 ldr imm,x
CC1F: 909 ldr imm,x
CC1F: 910 ldr imm,x
CC1F: 911 ldr imm,x
CC1F: 912 ldr imm,x
CC1F: 913 ldr imm,x
CC1F: 914 ldr imm,x
CC1F: 915 ldr imm,x
CC1F: 916 ldr imm,x
CC1F: 917 ldr imm,x
CC1F: 918 ldr imm,x
CC1F: 919 ldr imm,x
CC1F: 920 ldr imm,x
CC1F: 921 ldr imm,x
CC1F: 922 ldr imm,x
CC1F: 923 ldr imm,x
CC1F: 924 ldr imm,x
CC1F: 925 ldr imm,x
CC1F: 926 ldr imm,x
CC1F: 927 ldr imm,x
CC1F: 928 ldr imm,x
CC1F: 929 ldr imm,x
CC1F: 930 ldr imm,x
CC1F: 931 ldr imm,x
CC1F: 932 ldr imm,x
CC1F: 933 ldr imm,x
CC1F: 934 ldr imm,x
CC1F: 935 ldr imm,x
CC1F: 936 ldr imm,x
CC1F: 937 ldr imm,x
CC1F: 938 ldr imm,x
CC1F: 939 ldr imm,x

```

```

CC6F: 940 ldr imm,x
CC6F: 941 ldr imm,x
CC6F: 942 ldr imm,x
CC6F: 943 ldr imm,x
CC6F: 944 ldr imm,x
CC6F: 945 ldr imm,x
CC6F: 946 ldr imm,x
CC6F: 947 ldr imm,x
CC6F: 948 ldr imm,x
CC6F: 949 ldr imm,x
CC6F: 950 ldr imm,x
CC6F: 951 ldr imm,x
CC6F: 952 ldr imm,x
CC6F: 953 ldr imm,x
CC6F: 954 ldr imm,x
CC6F: 955 ldr imm,x
CC6F: 956 ldr imm,x
CC6F: 957 ldr imm,x
CC6F: 958 ldr imm,x
CC6F: 959 ldr imm,x
CC6F: 960 ldr imm,x
CC6F: 961 ldr imm,x
CC6F: 962 ldr imm,x
CC6F: 963 ldr imm,x
CC6F: 964 ldr imm,x
CC6F: 965 ldr imm,x
CC6F: 966 ldr imm,x
CC6F: 967 ldr imm,x
CC6F: 968 ldr imm,x
CC6F: 969 ldr imm,x
CC6F: 970 ldr imm,x
CC6F: 971 ldr imm,x
CC6F: 972 ldr imm,x
CC6F: 973 ldr imm,x
CC6F: 974 ldr imm,x
CC6F: 975 ldr imm,x
CC6F: 976 ldr imm,x
CC6F: 977 ldr imm,x
CC6F: 978 ldr imm,x
CC6F: 979 ldr imm,x
CC6F: 980 ldr imm,x
CC6F: 981 ldr imm,x
CC6F: 982 ldr imm,x
CC6F: 983 ldr imm,x
CC6F: 984 ldr imm,x
CC6F: 985 ldr imm,x
CC6F: 986 ldr imm,x
CC6F: 987 ldr imm,x
CC6F: 988 ldr imm,x
CC6F: 989 ldr imm,x
CC6F: 990 ldr imm,x
CC6F: 991 ldr imm,x
CC6F: 992 ldr imm,x
CC6F: 993 ldr imm,x
CC6F: 994 ldr imm,x
CC6F: 995 ldr imm,x
CC6F: 996 ldr imm,x
CC6F: 997 ldr imm,x
CC6F: 998 ldr imm,x
CC6F: 999 ldr imm,x

```

86 PC.CREAD	Set the IWM mode reg	29-JUL-85 15:19 PAGE 29	Set the IWM mode reg	86 PC.CREAD	Set the IWM mode reg	29-JUL-85 15:19 PAGE 30
61	ccbd:-85 48	sta checksum	61	CD1C:45 48	eor checksum	
62	ccbf:-c8	iny	62	CD1E:	CD1E:	
63	ccbf:-c8		63	CD1E:	CD1E:	
64	ccbf:-c8		64	CD1E:	CD1E:	
65	ccbf:-c8		65	CD1E:	CD1E:	
66	ccbf:-c8		66	CD1E:	CD1E:	
67	ccbf:-c8		67	CD1E:	CD1E:	
68	ccbf:-c8		68	CD1E:	CD1E:	
69	ccbf:-c8		69	CD1E:	CD1E:	
70	ccbf:-c8		70	CD1E:	CD1E:	
71	ccbf:-c8		71	CD1E:	CD1E:	
72	ccbf:-c8		72	CD1E:	CD1E:	
73	ccbf:-c8		73	CD1E:	CD1E:	
74	ccbf:-c8		74	CD1E:	CD1E:	
75	ccbf:-c8		75	CD1E:	CD1E:	
76	ccbf:-c8		76	CD1E:	CD1E:	
77	ccbf:-c8		77	CD1E:	CD1E:	
78	ccbf:-c8		78	CD1E:	CD1E:	
79	ccbf:-c8		79	CD1E:	CD1E:	
80	ccbf:-c8		80	CD1E:	CD1E:	
81	ccbf:-c8		81	CD1E:	CD1E:	
82	ccbf:-c8		82	CD1E:	CD1E:	
83	ccbf:-c8		83	CD1E:	CD1E:	
84	ccbf:-c8		84	CD1E:	CD1E:	
85	ccbf:-c8		85	CD1E:	CD1E:	
86	ccbf:-c8		86	CD1E:	CD1E:	
87	ccbf:-c8		87	CD1E:	CD1E:	
88	ccbf:-c8		88	CD1E:	CD1E:	
89	ccbf:-c8		89	CD1E:	CD1E:	
90	ccbf:-c8		90	CD1E:	CD1E:	
91	ccbf:-c8		91	CD1E:	CD1E:	
92	ccbf:-c8		92	CD1E:	CD1E:	
93	ccbf:-c8		93	CD1E:	CD1E:	
94	ccbf:-c8		94	CD1E:	CD1E:	
95	ccbf:-c8		95	CD1E:	CD1E:	
96	ccbf:-c8		96	CD1E:	CD1E:	
97	ccbf:-c8		97	CD1E:	CD1E:	
98	ccbf:-c8		98	CD1E:	CD1E:	
99	ccbf:-c8		99	CD1E:	CD1E:	
100	ccbf:-c8		100	CD1E:	CD1E:	
101	ccbf:-c8		101	CD1E:	CD1E:	
102	ccbf:-c8		102	CD1E:	CD1E:	
103	ccbf:-c8		103	CD1E:	CD1E:	
104	ccbf:-c8		104	CD1E:	CD1E:	
105	ccbf:-c8		105	CD1E:	CD1E:	
106	ccbf:-c8		106	CD1E:	CD1E:	
107	ccbf:-c8		107	CD1E:	CD1E:	
108	ccbf:-c8		108	CD1E:	CD1E:	
109	ccbf:-c8		109	CD1E:	CD1E:	
110	ccbf:-c8		110	CD1E:	CD1E:	
111	ccbf:-c8		111	CD1E:	CD1E:	
112	ccbf:-c8		112	CD1E:	CD1E:	
113	ccbf:-c8		113	CD1E:	CD1E:	
114	ccbf:-c8		114	CD1E:	CD1E:	
115	ccbf:-c8		115	CD1E:	CD1E:	
116	ccbf:-c8		116	CD1E:	CD1E:	
117	ccbf:-c8		117	CD1E:	CD1E:	
118	ccbf:-c8		118	CD1E:	CD1E:	
119	ccbf:-c8		119	CD1E:	CD1E:	
120	ccbf:-c8		120	CD1E:	CD1E:	

[illegible]

```

CD4B: 2 * equ
CD4B: 3 * bcc bentry
CD4B: 4 Entry
CD4B: 5 jmp bootcode
CD5B: 6 *
CD5B: 7 * X is still set to slot number.
CD5B: 8 *
CD5B: 9 *
CD5B: 10 bentry equ *
CD5B: 11 *
CD5B: 12 * lld lld*ROM
CD5B: 13 * lld lld*ROM
CD5B: 14 * lld lld*ROM
CD5B: 15 * lld lld*ROM
CD5B: 16 lentry equ *
CD5B: 17 *
CD5B: 18 *
CD5B: 19 *
CD5B: 20 *
CD5B: 21 *
CD5B: 22 *
CD5B: 23 * If this is a PC call, then get the address of the parm table
CD5B: 24 *
CD5B: 25 * lld ProfLeg.Y
CD5B: 26 * bni noplay
CD5B: 27 *
CD5B: 28 *
CD5B: 29 *
CD5B: 30 *
CD5B: 31 *
CD5B: 32 *
CD5B: 33 *
CD5B: 34 *
CD5B: 35 *
CD5B: 36 *
CD5B: 37 *
CD5B: 38 *
CD5B: 39 *
CD5B: 40 *
CD5B: 41 *
CD5B: 42 *
CD5B: 43 *
CD5B: 44 *
CD5B: 45 *
CD5B: 46 *
CD5B: 47 *
CD5B: 48 *
CD5B: 49 *
CD5B: 50 *
CD5B: 51 *
CD5B: 52 *
CD5B: 53 *
CD5B: 54 *
CD5B: 55 *
CD5B: 56 *
CD5B: 57 *
CD5B: 58 *
CD5B: 59 *
CD5B: 60 *
CD5B: 61 *
CD5B: 62 *
CD5B: 63 *
CD5B: 64 *
CD5B: 65 *
CD5B: 66 *
CD5B: 67 *
CD5B: 68 *
CD5B: 69 *
CD5B: 70 *
CD5B: 71 *
CD5B: 72 *
CD5B: 73 *
CD5B: 74 *
CD5B: 75 *
CD5B: 76 *
CD5B: 77 *
CD5B: 78 *
CD5B: 79 *
CD5B: 80 *
CD5B: 81 *
CD5B: 82 *
CD5B: 83 *
CD5B: 84 *

```

```

CD7D: 85 allset equ *
CD7D: 86 lld
CD7D: 87 *
CD7D: 88 *
CD7D: 89 *
CD7D: 90 *
CD7D: 91 *
CD7D: 92 *
CD7D: 93 *
CD7D: 94 *
CD7D: 95 *
CD7D: 96 *
CD7D: 97 *
CD7D: 98 *
CD7D: 99 *
CD7D: 100 *
CD7D: 101 *
CD7D: 102 *
CD7D: 103 *
CD7D: 104 *
CD7D: 105 *
CD7D: 106 *
CD7D: 107 *
CD7D: 108 *
CD7D: 109 *
CD7D: 110 *
CD7D: 111 *
CD7D: 112 *
CD7D: 113 *
CD7D: 114 *
CD7D: 115 *
CD7D: 116 *
CD7D: 117 *
CD7D: 118 *
CD7D: 119 *
CD7D: 120 *
CD7D: 121 *
CD7D: 122 *
CD7D: 123 *
CD7D: 124 *
CD7D: 125 *
CD7D: 126 *
CD7D: 127 *
CD7D: 128 *
CD7D: 129 *
CD7D: 130 *
CD7D: 131 *
CD7D: 132 *
CD7D: 133 *
CD7D: 134 *
CD7D: 135 *
CD7D: 136 *
CD7D: 137 *
CD7D: 138 *
CD7D: 139 *
CD7D: 140 *
CD7D: 141 *
CD7D: 142 *
CD7D: 143 *
CD7D: 144 *
CD7D: 145 *
CD7D: 146 *
CD7D: 147 *
CD7D: 148 *
CD7D: 149 *
CD7D: 150 *
CD7D: 151 *
CD7D: 152 *
CD7D: 153 *
CD7D: 154 *
CD7D: 155 *

```

```

CD7D: 156 *
CD7D: 157 *
CD7D: 158 *
CD7D: 159 *
CD7D: 160 *
CD7D: 161 *
CD7D: 162 *
CD7D: 163 *
CD7D: 164 *
CD7D: 165 *
CD7D: 166 *
CD7D: 167 *
CD7D: 168 *
CD7D: 169 *
CD7D: 170 *
CD7D: 171 *
CD7D: 172 *
CD7D: 173 *
CD7D: 174 *
CD7D: 175 *
CD7D: 176 *
CD7D: 177 *
CD7D: 178 *
CD7D: 179 *
CD7D: 180 *
CD7D: 181 *
CD7D: 182 *
CD7D: 183 *
CD7D: 184 *
CD7D: 185 *
CD7D: 186 *
CD7D: 187 *
CD7D: 188 *
CD7D: 189 *
CD7D: 190 *
CD7D: 191 *
CD7D: 192 *
CD7D: 193 *
CD7D: 194 *
CD7D: 195 *
CD7D: 196 *
CD7D: 197 *
CD7D: 198 *
CD7D: 199 *
CD7D: 200 *

```

```

156 * CDD1: ldx CMDUnit
157 * CDD1:A5 43 bne skipcopy ;Never mind
158 * CDD3:D0 6A CE3F
159 * CDD5:
160 * Check the parameter count for this call to unit#0
161 *
162 * CDD5:A6 42 ldx CMDCode
163 * CDD7:BD 06 CF ldx permctab,
164 * CDDA:29 7F ldx #0
165 * CDDC:A8 tdy
166 * CDDD:A9 04 ldx #BadPcnt
167 * CDDF:C4 5A cpy Unit
168 * CDE1:D0 DB bne ErrorHitch
169 * CDE3:
170 * Now service one of the three commands
171 *
172 * CDE3:E0 05 cpx
173 * CDE5:D0 0A bne notinit
174 * CDE7:A9 06 CF ldx #PowerReset
175 * CDE9:AD 58 ldx #AlignID
176 * CDEB:AD 58 ldx #0
177 * CDEE:4C 31 CF jmp se2
178 * CDF1:
179 * CDF1:BD 24 CE10
180 * CDF4:D0 24 bne maybectrl
181 *
182 * CDF4:A9 21 ldx #BadCtl
183 * CDF6:A6 46 ldx CMDSCode
184 * CDF8:D0 C4 CDBE
185 * CDFB:0A bne ErrorHitch
186 * CDFC:0A tdx
187 * CDFB:A6 58 ldx #0
188 * CDFD:A0 07 ldy Slot
189 * CDFE:91 44 sta (CmdBuffer).y ;Clear some space
190 * CDFE:91 44 tdy
191 * CDFE:91 44 bne nin1
192 * CDFE:91 44 ldx NumDevices.x
193 * CDFE:91 44 sta (CmdBuffer).y ;Stick it where they want it
194 * CDFE:91 44 iny
195 * CDFE:91 44 ldx #0
196 * CDEA:
197 * CDEA:AD F9 04 ldx #4F9
198 * CDEB:
199 * CDEB:
200 * CDEB:
201 * CDEB:
202 * CDEB:
203 * CDEB:91 44 sta (CmdBuffer).y ;Store PC interrupt status
204 * CDEB:A9 08 ldx #0
205 * CDEB:AD 58 ldx #0
206 * CDEB:AD 58 ldx #0
207 * CDEB:AD 58 ldx #0
208 * CDEB:AD 58 ldx #0
209 * CDEB:AD 58 ldx #0
210 * CDEB:AD 58 ldx #0
211 * CDEB:AD 58 ldx #0
212 * CDEB:AD 58 ldx #0
213 * CDEB:AD 58 ldx #0
214 * CDEB:AD 58 ldx #0
215 * CDEB:AD 58 ldx #0
216 * CDEB:AD 58 ldx #0
217 * CDEB:AD 58 ldx #0
218 * CDEB:AD 58 ldx #0
219 * CDEB:AD 58 ldx #0
220 * CDEB:AD 58 ldx #0
221 * CDEB:AD 58 ldx #0
222 * CDEB:AD 58 ldx #0
223 * CDEB:AD 58 ldx #0
224 * CDEB:AD 58 ldx #0
225 * CDEB:AD 58 ldx #0
226 * CDEB:AD 58 ldx #0
227 * CDEB:AD 58 ldx #0

```

```

228 * CDEB:A9 C0 ldx #0
229 * CDEB:BD F9 05 ldx #5F9
230 * CDEB:A9 0F ldx #0
231 * CDEB:BD 05 CE3C
232 * CDEB:BD 05 CE3C
233 * CDEB:BD 05 CE3C
234 * CDEB:BD 05 CE3C
235 * CDEB:BD 05 CE3C
236 * CDEB:BD 05 CE3C
237 * CDEB:BD 05 CE3C
238 * CDEB:BD 05 CE3C
239 * CDEB:BD 05 CE3C
240 * CDEB:BD 05 CE3C
241 * CDEB:BD 05 CE3C
242 * CDEB:BD 05 CE3C
243 * CDEB:BD 05 CE3C
244 * CDEB:BD 05 CE3C
245 * CDEB:BD 05 CE3C
246 * CDEB:BD 05 CE3C
247 * CDEB:BD 05 CE3C
248 * CDEB:BD 05 CE3C
249 * CDEB:BD 05 CE3C
250 * CDEB:BD 05 CE3C
251 * CDEB:BD 05 CE3C
252 * CDEB:BD 05 CE3C
253 * CDEB:BD 05 CE3C
254 * CDEB:BD 05 CE3C
255 * CDEB:BD 05 CE3C
256 * CDEB:BD 05 CE3C
257 * CDEB:BD 05 CE3C
258 * CDEB:BD 05 CE3C
259 * CDEB:BD 05 CE3C
260 * CDEB:BD 05 CE3C
261 * CDEB:BD 05 CE3C
262 * CDEB:BD 05 CE3C
263 * CDEB:BD 05 CE3C
264 * CDEB:BD 05 CE3C
265 * CDEB:BD 05 CE3C
266 * CDEB:BD 05 CE3C
267 * CDEB:BD 05 CE3C
268 * CDEB:BD 05 CE3C
269 * CDEB:BD 05 CE3C
270 * CDEB:BD 05 CE3C
271 * CDEB:BD 05 CE3C
272 * CDEB:BD 05 CE3C
273 * CDEB:BD 05 CE3C
274 * CDEB:BD 05 CE3C
275 * CDEB:BD 05 CE3C
276 * CDEB:BD 05 CE3C
277 * CDEB:BD 05 CE3C
278 * CDEB:BD 05 CE3C
279 * CDEB:BD 05 CE3C
280 * CDEB:BD 05 CE3C
281 * CDEB:BD 05 CE3C
282 * CDEB:BD 05 CE3C
283 * CDEB:BD 05 CE3C
284 * CDEB:BD 05 CE3C
285 * CDEB:BD 05 CE3C
286 * CDEB:BD 05 CE3C
287 * CDEB:BD 05 CE3C
288 * CDEB:BD 05 CE3C
289 * CDEB:BD 05 CE3C
290 * CDEB:BD 05 CE3C
291 * CDEB:BD 05 CE3C
292 * CDEB:BD 05 CE3C
293 * CDEB:BD 05 CE3C
294 * CDEB:BD 05 CE3C
295 * CDEB:BD 05 CE3C
296 * CDEB:BD 05 CE3C
297 * CDEB:BD 05 CE3C
298 * CDEB:BD 05 CE3C
299 * CDEB:BD 05 CE3C
300 * CDEB:BD 05 CE3C
301 * CDEB:BD 05 CE3C
302 * CDEB:BD 05 CE3C

```

87 PC.MAIN Protocol Converter / CBus Driver 29-JUL-85 15:19 PAGE 35

```

CEB1: 383 *
CEB1:20 EC CA 384 * jar SendPack
CEB1:B0 46 CEEC 385 * bcs behitch
CEB1: 386 *
CEB1: 387 * Now copy over the buffer address for any data xfer..
CEB1: 388 *
CEB1:A5 44 CEEC 389 * ldx CMDBuffer
CEB1:54 44 CEEC 390 * sta buffer
CEB1:55 44 CEEC 391 * ldx buffer+1
CEB1: 392 *
CEB1: 393 * Now for some commands, we have to send over a packet of data,
CEB1: 394 * too..
CEB1: 395 * See if this command is one of THOSE.
CEB1: 396 *
CEB1:A6 42 CEEC 397 * ldx endcode
CEB1:B0 66 CF CEEC 398 * ldx permctab,x
CEB1:10 3B CEEC 399 * bpl noxtresend
CEB1: 400 *
CEB1: 401 * The buffer address and bytecount depend on the call type.
CEB1: 402 *
CEB1: 403 *
CEB1:E0 84 CEEC 404 * cpx #ControlCmd
CEB1:D0 1B CEEC 405 * bne NOControl
CEB1: 406 *
CEB1: 407 * In the case of control, bytecount:=(buffer) then
CEB1: 408 * buffer:=buffer+2
CEB1: 409 *
CEB1: 410 *
CEB1: 411 *
CEB1: 412 *
CEB1: 413 *
CEB1: 414 *
CEB1: 415 *
CEB1: 416 *
CEB1: 417 *
CEB1: 418 *
CEB1: 419 *
CEB1: 420 *
CEB1: 421 *
CEB1: 422 *
CEB1: 423 *
CEB1: 424 *
CEB1: 425 *
CEB1: 426 *
CEB1: 427 *
CEB1: 428 *
CEB1: 429 *
CEB1: 430 *
CEB1: 431 *
CEB1: 432 *
CEB1: 433 *
CEB1: 434 *
CEB1: 435 *
CEB1: 436 *
CEB1: 437 *
CEB1: 438 *
CEB1: 439 *
CEB1: 440 *
CEB1: 441 *
CEB1: 442 *
CEB1: 443 *
CEB1: 444 *

```

87 PC.MAIN Protocol Converter / CBus Driver 29-JUL-85 15:19 PAGE 36

```

CEB1: 374 * On ProDOS status call, we've got to point the buffer pointer
CEB1: 375 * correctly to zero page...it's the only case special case
CEB1: 376 * (On Write, Format and Control no data comes back).
CEB1: 377 *
CEB1: 378 *
CEB1: 379 *
CEB1: 380 *
CEB1: 381 *
CEB1: 382 *
CEB1: 383 *
CEB1: 384 *
CEB1: 385 *
CEB1: 386 *
CEB1: 387 *
CEB1: 388 *
CEB1: 389 *
CEB1: 390 *
CEB1: 391 *
CEB1: 392 *
CEB1: 393 *
CEB1: 394 *
CEB1: 395 *
CEB1: 396 *
CEB1: 397 *
CEB1: 398 *
CEB1: 399 *
CEB1: 400 *
CEB1: 401 *
CEB1: 402 *
CEB1: 403 *
CEB1: 404 *
CEB1: 405 *
CEB1: 406 *
CEB1: 407 *
CEB1: 408 *
CEB1: 409 *
CEB1: 410 *
CEB1: 411 *
CEB1: 412 *
CEB1: 413 *
CEB1: 414 *
CEB1: 415 *
CEB1: 416 *
CEB1: 417 *
CEB1: 418 *
CEB1: 419 *
CEB1: 420 *
CEB1: 421 *
CEB1: 422 *
CEB1: 423 *
CEB1: 424 *
CEB1: 425 *
CEB1: 426 *
CEB1: 427 *
CEB1: 428 *
CEB1: 429 *
CEB1: 430 *
CEB1: 431 *
CEB1: 432 *
CEB1: 433 *
CEB1: 434 *
CEB1: 435 *
CEB1: 436 *
CEB1: 437 *
CEB1: 438 *
CEB1: 439 *
CEB1: 440 *
CEB1: 441 *
CEB1: 442 *
CEB1: 443 *
CEB1: 444 *

```



```
CF98: 547 * CF98 AssignID equ
CF99: 548 resetchain pha
CF9A: 549 !Save the init code things
CF9B: 550 !Reset all of those things
CF9C: 551 !Save InitCode
CF9D: 552 !Save InitCode
CF9E: 553 * Save the command code, unit, and init code 'cause we'll trample
CF9F: 554 * 'em.
CF9G: 555 !da CMDCode
CF9H: 556 !pha: CMDPCount
CF9I: 557 !pha: CMDPCount
CF9J: 558 !pha: CMDPCount
CF9K: 559 !pha: CMDPCount
CF9L: 560 !pha: CMDPCount
CF9M: 561 !pha: CMDPCount
CF9N: 562 !pha: CMDPCount
CF9O: 563 * Store away the type of INIT
CF9P: 564 *
CF9Q: 565 * Set up to send DefID command packets
CF9R: 566 *
CF9S: 567 !da: #InitCmd
CF9T: 568 !da: #InitCmd
CF9U: 569 !da: #Unit
CF9V: 570 !da: #Unit
CF9W: 571 !da: #Unit
CF9X: 572 !da: #Unit
CF9Y: 573 * Point the buffer pointer
CF9Z: 574 *
CF9A0: 575 !da: #CMDCode
CF9A1: 576 !da: #CMDCode
CF9A2: 577 !da: #CMDCode
CF9A3: 578 !da: #CMDCode
CF9A4: 579 !da: #CMDCode
CF9A5: 580 !da: #CMDCode
CF9A6: 581 *
CF9A7: 582 *
CF9A8: 583 *
CF9A9: 584 *
CF9AA: 585 *
CF9AB: 586 *
CF9AC: 587 *
CF9AD: 588 *
CF9AE: 589 *
CF9AF: 590 *
CF9AG: 591 *
CF9AH: 592 *
CF9AI: 593 *
CF9AJ: 594 *
CF9AK: 595 *
CF9AL: 596 *
CF9AM: 597 *
CF9AN: 598 *
CF9AO: 599 *
CF9AP: 600 *
CF9AQ: 601 *
CF9AR: 602 *
CF9AS: 603 *
CF9AT: 604 *
CF9AU: 605 *
CF9AV: 606 *
CF9AW: 607 *
CF9AX: 608 *
CF9AY: 609 *
CF9AZ: 610 *
CF9B0: 611 *
CF9B1: 612 *
CF9B2: 613 *
CF9B3: 614 *
CF9B4: 615 *
CF9B5: 616 *
CF9B6: 617 *
```

```
CFE7: 618 !feq IIC*ROM
CFE8: 619 !fin
CFE9: 620 !fin
CFEA: 621 !fin
CFEB: 622 !fin
CFEC: 623 !fin
CFED: 624 !fin
CFEE: 625 !fin
CFEF: 626 !fin
CFEG: 627 !fin
CFEH: 628 !fin
CFEI: 629 !fin
CFEJ: 630 !fin
CFEK: 631 !fin
CFEL: 632 !fin
CFEM: 633 !fin
CFEN: 634 !fin
CFEO: 635 !fin
CFEP: 636 !fin
CFEQ: 637 !fin
CFER: 638 !fin
CFES: 639 !fin
CFET: 640 !fin
CFEU: 641 !fin
CFEV: 642 !fin
CFEW: 643 !fin
CFEX: 644 !fin
CFEY: 645 !fin
CFEZ: 646 !fin
CFEA0: 647 !fin
CFEA1: 648 !fin
CFEA2: 649 !fin
CFEA3: 650 !fin
CFEA4: 651 !fin
CFEA5: 652 !fin
CFEA6: 653 !fin
CFEA7: 654 !fin
CFEA8: 655 !fin
CFEA9: 656 !fin
CFEAA: 657 !fin
CFEAB: 658 !fin
CFEAC: 659 !fin
CFEAD: 660 !fin
CFEAE: 661 !fin
CFEAF: 662 !fin
CFEAG: 663 !fin
CFEAH: 664 !fin
CFEAI: 665 !fin
CFEAJ: 666 !fin
CFEAK: 667 !fin
CFEAL: 668 !fin
CFEAM: 669 !fin
CFEAN: 670 !fin
CFEAO: 671 !fin
CFEAP: 672 !fin
CFEAP: 673 !fin
CFEAP: 674 !fin
CFEAP: 675 !fin
CFEAP: 676 !fin
CFEAP: 677 !fin
CFEAP: 678 !fin
CFEAP: 679 !fin
CFEAP: 680 !fin
CFEAP: 681 !fin
CFEAP: 682 !fin
CFEAP: 683 !fin
```



```

SOURCE FILE #01 =>APTALK.2C
INCLUDE FILE #02 =>APPLETALK2C.D1/APTALK.VARIABLES
INCLUDE FILE #03 =>APPLETALK2C.D2/APTALK.C700
INCLUDE FILE #04 =>APPLETALK2C.D2/APTALK.R0MSTUFF
0000: 0001 GENERAL EQU 0
0000: 0002 INCD0BX EQU 0
0000: 4 *****
0000: 5 *****
0000: 6 *****
0000: 7 *****
0000: 8 *****
0000: 9 *****
0000: 10 *****
0000: 11 *****
0000: 12 *****
0000: 13 *****
0000: 14 *****
0000: 15 *****
0000: 16 *****
0000: 18 * This file contains the includes necessary to
0000: 19 * generate the AppleTalk //c code for the converter
0000: 20 * 50x and the code to be placed in the //c ROM.
0000:
0000: 0002 MSB ON
0000: 23 X6502 ON
0000: 25 LST NCYCLES
0000: 26 INCLUDE /APPLETALK2C.D1/APTALK.VARIABLES

```

```

0000: 3 *****
0000: 4 *****
0000: 5 *****
0000: 6 *****
0000: 7 *****
0000: 8 *****
0000: 9 *****
0000: 10 *****
0000: 11 *****
0000: 12 *****
0000: 13 *****
0000: 14 *****
0000: 15 *****
0000: 192 LST ON
0000:
0000: 194 * Apple //c zero page used at boot and not restored.
0000: 0000 196 ZPR EQU #8 ;Used and not restored
0000:
0000: 198 * AppleTalk //c Converter Box stuff
0000:
0000: 200 *INITCMD EQU 1 ;init command #
0000: 201 *READCMD EQU 2 ;Read rest of pkt cmd #
0000: 202 *WRITECMD EQU 3 ;Write pkt command #
0000: 203 *STATCMD EQU 4 ;Stat pkt command #
0000: 204 *READPCMD EQU 5 ;Read protocol cmd #
0000: 0001 205 DIAGCMD EQU #81 ;Diag call command #
0000:
0000: 207 * The following table contains the only
0000: 208 * valid commands recognized by the converter
0000: 209 * AppleTalk//c box when using the protocol
0000: 210 * converter's STATUS command.
0000:
0000: 212 * 00-Short status request
0000: 0000 213 * 01-Return DCB info
0000: 0000 214 * 02-Return IDB
0000: 0000 215 * 03-Return IDB
0000: 0004 216 CMDCINIT EQU #4
0000: 0005 217 CMDSTATUS EQU #5
0000: 0006 218 CMDREADREST EQU #6
0000: 0007 219 CMDREADPRDT EQU #7
0000: 0008 220 CMDREADPRDT EQU #8
0000: 0009 221 CMDCREBOOT EQU #9
0000: 000A 222 CMDCIDT EQU #A
0000: 0000 223 * 0B-AppleTalk ID call 2
0000:
0000: 225 * Protocol converter commands
0000: 0000 227 PCSTATUSCMD EQU #0 ;Prot Conv status command
0000: 228 * 01-Prot Conv readblk command
0000: 0000 229 * 02-Prot Conv writeblk command
0000: 0000 230 * 03-Prot Conv format command
0000: 0000 231 * 04-Prot Conv init command
0000: 0000 232 * 05-Prot Conv init command
0000: 0000 233 * 06-Prot Conv open command
0000: 0000 234 * 07-Prot Conv close command
0000: 0000 235 * 08-Prot Conv read command
0000: 0000 236 PCWRITECMD EQU #5 ;Prot Conv write command
0000:
0000: 238 * RELVERNUM is the version number
0000: 239 * for 65C02 RELEASE VERSION NUMBER.
0000: 0000 240 * It must be kept updated as this product
0000: 0000 241 * is updated.
0000: 0000 243 RELVERNUM EQU 0 ;Release version 0-0

```

```

;no list/list general stuff
;DC0M stuff only/both ROM+box
*****
AppleTalk //c
INCLUDES File
*****
by
Fern Bachman
Copyright Apple Computer, Inc. 1985
All Rights Reserved.
*****

```

```

0000:
0000: 245 * STATBYTE codes
0000: 247 CMDERROR EQU $01*800 ;Illegal Command Type
0000: 248 BADUNITR EQU $11*800 ;Bad unit number.
0000: 249 BADPCNTR EQU $22*800 ;Bad packet error.
0000: 250 BADPCNT EQU $22*800 ;Bad packet error.
0000: 251 NODEVCN EQU $28*800 ;Dev to access not connected
0000:
0000: 253 * LSTNONBT indicates this device is the last
0000: 254 * one in the chain and if it is also
0000: 255 * unit number then it is not a
0000: 256 * bootable device.
0000:
0000: 258 *LSTERRBT EQU $48*800 ;last one in chain and bootable
0000: 259 LSTNONBT EQU $41*800 ;last one in chain/NOT bootable
0000:
0000: 261 * AppleTalk specific error codes for STATBYTE.
0000:
0000: 263 NOKTINBUF EQU $88*838*801 ;No packets in buffer
0000: 264 BUFFEROVER EQU $88*838*802 ;Buffer overflowed
0000: 265 READPSTEND EQU $88*838*803 ;Tried to read past end of pkt
0000: 266 NOUNIQUEID EQU $88*838*804 ;No unique node addr. found
0000: 267 EXCESSDEF EQU $88*838*805 ;Too many collisions
0000: 268 BYTES2SEND EQU $88*838*806 ;# bytes to send > 255
0000: 269 EXCESSDEFS EQU $88*838*807 ;Excessive deferrals
0000: 270 TOMANYCOL EQU $88*838*808 ;Too many collisions
0000: 271 ILLAPTYPE EQU $88*838*809 ;Illegal lrp type (> $7F)
0000: 272 NOLAPTYPE EQU $88*838*80A ;No LAP type
0000: 273 NOLAPTYPE EQU $88*838*80B ;No LAP type
0000: 274 ID1 EQU $F ;ID byte1 for finding APTalk
0000: 275 ID2 EQU $B ;ID byte2 for finding APTalk
0000:
0000: 277 PCOUNTG EQU $3 ;Control call PCOUNT
0000: 278 PCOUNTI EQU $4 ;Init call PCOUNT
0000: 279 PCOUNTI EQU $2 ;Init call PCOUNT
0000: 280 PCOUNTS4.3 EQU $3 ;Status call PCOUNT for 4-B
0000: 281 PCOUNTS4.B EQU $6 ;Status call PCOUNT for 4-B
0000:
0000: 285 LST DN
0000:
0000: 287 * Apple //c zero page usage
0000:
0000: 289 KSMH EQU $39 ;Input hook hi byte
0000:
0000: 291 DSECT
0000: 292 DRG EQU $00
0000: 293 ZP2CUSE EQU *
0000: 294 PARAMNUM DS 1,0 ;Number of parameters
0000: 295 PARAMNUM DS 1,0 ;Number of parameters
0000: 296 PTRBUF DS 2,0 ;Buffer pointer
0000: 297 CODECHDS EQU * ;Command code
0000: 298 NUMLOMRITE DS 1,0 ;# of bytes to write to box
0000: 299 BYTELONUM EQU * ;# of bytes to read from box
0000: 300 NUMHWRITE EQU * ;# of bytes to write hi byte
0000: 301 BYTELONUM DS 1,0 ;# of bytes to read from box
0000: 302 BYTELONUM DS 1,0 ;# of bytes to read from box
0000: 303 TYPEWRITE DS 1,0 ;Write type code
0000: 304 TESTTMP DS 2,0 ;Diag read from address
0000:
0000: 306 ADDR0 DS 1,0 ;Used as temp and restored
0000: 307 ADDR1 DS 1,0 ;Used as temp and restored
0000:
0000: 309 ZP2CUSELEN EQU *-ZP2CUSE ;# of bytes used in //c zpage
0000:
0000: 311 BOOTIT DS 2,0 ;Boot prog strt adr

```

```

02 APTALK.VARIABLES AppleTalk //c Variables.
0000: 313 DEND
0000: 315 * AppleTalk //c non zero page usage

```

```

0000: 03F2 317 SOFTEV EQU $3F2 ;Reset vector
0000: 03F3 318 INIT EQU $3F3 ;Init screen hole
0000: 03F4 319 SCRNMH2 EQU $3F4 ;DRIVER #start-1
0000: 03F5 320 SCRNMH1 EQU $3F5 ;Print drv #start-1
0000: 03F6 321 SCRNMH2 EQU $3F6 ;Print drv #start-1
0000: 03F7 322 SCRNMH1 EQU $3F7 ;Temp use only
0000: 03F8 323 WSLDT EQU $3F8 ;Card slot # (6ch) for ints
0000: 03F9 324 ALTRNSW EQU $3F9 ;Switch to alt ROM vector
0000: 03FA 325 MAINROMSW EQU $3FA ;Switch to main ROM vector
0000: 03FB 326 MAINROMSW EQU $3FB ;Get a Trn mode
0000: 03FC 327 INIT EQU $3FC ;Get a Trn mode
0000: 03FD 328 HOME EQU $3FD ;Clear screen
0000: 03FE 329 SETNORM EQU $3FE ;Normal char display
0000: 03FF 330 SETKBD EQU $3FF ;Keyboard is input device
0000: 0400 331 SEIVID EQU $400 ;Video is output device
0000:
0000: 333 ELSE
0000: 334 FIN
0000:
0000: 335 LST DN
0000:
0000: 336 DO GENERAL
0000:
0000: 337 LST DN
0000:
0000: 338 DRG EQU $C700 ;Cn88 page for //c goes here
0000: 339 C700 EQU $C700 ;Cn88 page for //c goes here
0000: 340 INCLUDE /APPLETALK2C.D2/APTALK.C700

```

```

C788: .....
C789: .....
C790: .....
C791: .....
C792: .....
C793: .....
C794: .....
C795: .....
C796: .....
C797: .....
C798: .....
C799: .....
C800: .....
C801: .....
C802: .....
C803: .....
C804: .....
C805: .....
C806: .....
C807: .....
C808: .....
C809: .....
C810: .....
C811: .....
C812: .....
C813: .....
C814: .....
C815: .....
C816: .....
C817: .....
C818: .....
C819: .....
C820: .....
C821: .....
C822: .....
C823: .....
C824: .....
C825: .....
C826: .....
C827: .....
C828: .....
C829: .....
C830: .....
C831: .....
C832: .....
C833: .....
C834: .....
C835: .....
C836: .....
C837: .....
C838: .....
C839: .....
C840: .....
C841: .....
C842: .....
C843: .....
C844: .....
C845: .....
C846: .....
C847: .....
C848: .....
C849: .....
C850: .....
C851: .....
C852: .....
C853: .....
C854: .....
C855: .....
C856: .....
C857: .....
C858: .....
C859: .....
C860: .....
C861: .....
C862: .....
C863: .....
C864: .....
C865: .....
C866: .....
C867: .....
C868: .....
C869: .....
C870: .....
C871: .....
C872: .....
C873: .....
C874: .....
C875: .....
C876: .....
C877: .....
C878: .....
C879: .....
C880: .....
C881: .....
C882: .....
C883: .....
C884: .....
C885: .....
C886: .....
C887: .....
C888: .....
C889: .....
C890: .....
C891: .....
C892: .....
C893: .....
C894: .....
C895: .....
C896: .....
C897: .....
C898: .....
C899: .....
C900: .....
C901: .....
C902: .....
C903: .....
C904: .....
C905: .....
C906: .....
C907: .....
C908: .....
C909: .....
C910: .....
C911: .....
C912: .....
C913: .....
C914: .....
C915: .....
C916: .....
C917: .....
C918: .....
C919: .....
C920: .....
C921: .....
C922: .....
C923: .....
C924: .....
C925: .....
C926: .....
C927: .....
C928: .....
C929: .....
C930: .....
C931: .....
C932: .....
C933: .....
C934: .....
C935: .....
C936: .....
C937: .....
C938: .....
C939: .....
C940: .....
C941: .....
C942: .....
C943: .....
C944: .....
C945: .....
C946: .....
C947: .....
C948: .....
C949: .....
C950: .....
C951: .....
C952: .....
C953: .....
C954: .....
C955: .....
C956: .....
C957: .....
C958: .....
C959: .....
C960: .....
C961: .....
C962: .....
C963: .....
C964: .....
C965: .....
C966: .....
C967: .....
C968: .....
C969: .....
C970: .....
C971: .....
C972: .....
C973: .....
C974: .....
C975: .....
C976: .....
C977: .....
C978: .....
C979: .....
C980: .....
C981: .....
C982: .....
C983: .....
C984: .....
C985: .....
C986: .....
C987: .....
C988: .....
C989: .....
C990: .....
C991: .....
C992: .....
C993: .....
C994: .....
C995: .....
C996: .....
C997: .....
C998: .....
C999: .....

```

```

C712: * The entry point APPTALK must appear at
C713: * $Cn2 in this and all future Appletalk cards
C714: * for the Apple // product line.
C715:
C716:
C717:
C718:
C719:
C720:
C721:
C722:
C723:
C724:
C725:
C726:
C727:
C728:
C729:
C730:
C731:
C732:
C733:
C734:
C735:
C736:
C737:
C738:
C739:
C740:
C741:
C742:
C743:
C744:
C745:
C746:
C747:
C748:
C749:
C750:
C751:
C752:
C753:
C754:
C755:
C756:
C757:
C758:
C759:
C760:
C761:
C762:
C763:
C764:
C765:
C766:
C767:
C768:
C769:
C770:
C771:
C772:
C773:
C774:
C775:
C776:
C777:
C778:
C779:
C780:
C781:
C782:
C783:
C784:
C785:
C786:
C787:
C788:
C789:
C790:
C791:
C792:
C793:
C794:
C795:
C796:
C797:
C798:
C799:
C800:
C801:
C802:
C803:
C804:
C805:
C806:
C807:
C808:
C809:
C810:
C811:
C812:
C813:
C814:
C815:
C816:
C817:
C818:
C819:
C820:
C821:
C822:
C823:
C824:
C825:
C826:
C827:
C828:
C829:
C830:
C831:
C832:
C833:
C834:
C835:
C836:
C837:
C838:
C839:
C840:
C841:
C842:
C843:
C844:
C845:
C846:
C847:
C848:
C849:
C850:
C851:
C852:
C853:
C854:
C855:
C856:
C857:
C858:
C859:
C860:
C861:
C862:
C863:
C864:
C865:
C866:
C867:
C868:
C869:
C870:
C871:
C872:
C873:
C874:
C875:
C876:
C877:
C878:
C879:
C880:
C881:
C882:
C883:
C884:
C885:
C886:
C887:
C888:
C889:
C890:
C891:
C892:
C893:
C894:
C895:
C896:
C897:
C898:
C899:
C900:
C901:
C902:
C903:
C904:
C905:
C906:
C907:
C908:
C909:
C910:
C911:
C912:
C913:
C914:
C915:
C916:
C917:
C918:
C919:
C920:
C921:
C922:
C923:
C924:
C925:
C926:
C927:
C928:
C929:
C930:
C931:
C932:
C933:
C934:
C935:
C936:
C937:
C938:
C939:
C940:
C941:
C942:
C943:
C944:
C945:
C946:
C947:
C948:
C949:
C950:
C951:
C952:
C953:
C954:
C955:
C956:
C957:
C958:
C959:
C960:
C961:
C962:
C963:
C964:
C965:
C966:
C967:
C968:
C969:
C970:
C971:
C972:
C973:
C974:
C975:
C976:
C977:
C978:
C979:
C980:
C981:
C982:
C983:
C984:
C985:
C986:
C987:
C988:
C989:
C990:
C991:
C992:
C993:
C994:
C995:
C996:
C997:
C998:
C999:

```

```

83 APTALK.C700 AppleTalk//c C700 Rins 29-JUL-85 18:33 PAGE 7
C748:08 83 C74D 148 BCS APTALKOFFLN :Error so display message
C74A:8C 88 141 JMP (BOOTITT) :Start of boot code
C74D:AD 81 C74D 143 APTALKOFFLN EQU * :Switch in LC ROM
C74E:AD 81 C74D 144 LDA AC0B1 :No inverse stuff
C750:AD 81 C74D 145 LDA AC0B1 :No inverse stuff
C753:20 84 FE 146 JSR SETNORM :Fix up some stuff
C756:20 2F FB 147 JSR INIT :Clear screen for message
C759:20 58 FC 148 JSR HOME :Screen is output device
C75C:20 93 FE 149 JSR SETVID :Keyboard is input device
C75F:20 89 FE 150 JSR SETKBD :Keyboard is input device
C762:08 18 152 LDY *APOFFMSGLN-1 :length of error message
C764:08 6F C764 153 APOFFLOOP EQU * :Get character to show
C766:08 6F C764 154 LDA APOFFMSG,Y :Display on screen
C767:99 DB 07 155 STA *7DB,Y :
C76A:08 156 DEY APOFFLOOP :
C76B:18 F7 C764 157 BRAHANGLOOP EQU BRAHANGLOOP :Loop till done
C76D:08 158 BRA BRAHANGLOOP :Loop forever
C76F: 161 MSB: DN
C76F:01 F8 F8 EC C76F 163 APOFFMSG EQU * :AppleTalk
C770:01 F8 F8 EC C76F 164 ASC *APOFFMSG :
C778: 8811 166 APOFFMSGLN EQU * :APOFFMSG :Length of error message
C778: 8800 168 C7FILL00 EQU *C778-0 :
C778: 8800 169 DS C7FILL00,$FF :Fill to version number
C77F: 171 DSECT :Version # goes at $C7FF
C77F: 172 ORG: $C7FF :Release version number
C77F: 174 *RELVERSION EQU * :
C77F: 175 DFB RELVERSION :
C77F: 00 DEND
C780: 177

```

```

-- NEXT OBJECT FILE NAME IS APTALK.1
C508: 32 ORG: $C508-$C700 in aux ROM
C509: 33 INCLUDE /APPLETALK2C.D2/APTALK.ROMSTUFF

```

```

84 APTALK.ROMSTUFF AppleTalk//c Alt-ROM stuff 29-JUL-85 18:33 PAGE 8
C508: 4 *****
C509: 5 AppleTalk //c Protocol Converter
C50A: 6 *****
C50B: 7 Alternate ROM Stuff Routines
C50C: 8 *****
C50D: 9 ***** by Fern Bachman
C50E: 10 ***** Copyright Apple Computer, Inc. 1985
C50F: 11 ***** All Rights Reserved.
C510: 12 *****
C511: 13 *****
C512: 14 *****
C513: 15 *****
C514: 16 *****
C515: 17 *****
C516: 18 *****
C517: 19 *****
C518: 20 *****
C519: 21 *****
C520: 22 *****
C521: 23 *****
C522: 24 *****
C523: 25 *****
C524: 26 *****
C525: 27 *****
C526: 28 *****
C527: 29 *****
C528: 30 *****
C529: 31 *****
C530: 32 *****
C531: 33 *****
C532: 34 *****
C533: 35 *****
C534: 36 *****
C535: 37 *****
C536: 38 *****
C537: 39 *****
C538: 40 *****
C539: 41 *****
C540: 42 *****
C541: 43 *****
C542: 44 *****
C543: 45 *****
C544: 46 *****
C545: 47 *****
C546: 48 *****
C547: 49 *****
C548: 50 *****
C549: 51 *****
C550: 52 *****
C551: 53 *****
C552: 54 *****
C553: 55 *****
C554: 56 *****
C555: 57 *****
C556: 58 *****
C557: 59 *****
C558: 60 *****
C559: 61 *****
C560: 62 *****
C561: 63 *****
C562: 64 *****
C563: 65 *****
C564: 66 *****
C565: 67 *****
C566: 68 *****
C567: 69 *****
C568: 70 *****
C569: 71 *****
C570: 72 *****
C571: 73 *****
C572: 74 *****
C573: 75 *****
C574: 76 *****
C575: 77 *****
C576: 78 *****
C577: 79 *****
C578: 80 *****
C579: 81 *****
C580: 82 *****
C581: 83 *****
C582: 84 *****
C583: 85 *****
C584: 86 *****
C585: 87 *****
C586: 88 *****
C587: 89 *****
C588: 90 *****
C589: 91 *****
C590: 92 *****
C591: 93 *****
C592: 94 *****
C593: 95 *****
C594: 96 *****
C595: 97 *****
C596: 98 *****
C597: 99 *****
C598: 100 *****
C599: 101 *****
C600: 102 *****
C601: 103 *****
C602: 104 *****
C603: 105 *****
C604: 106 *****
C605: 107 *****
C606: 108 *****
C607: 109 *****
C608: 110 *****
C609: 111 *****
C610: 112 *****
C611: 113 *****
C612: 114 *****
C613: 115 *****
C614: 116 *****
C615: 117 *****
C616: 118 *****
C617: 119 *****
C618: 120 *****
C619: 121 *****
C620: 122 *****
C621: 123 *****
C622: 124 *****
C623: 125 *****
C624: 126 *****
C625: 127 *****
C626: 128 *****
C627: 129 *****
C628: 130 *****
C629: 131 *****
C630: 132 *****
C631: 133 *****
C632: 134 *****
C633: 135 *****
C634: 136 *****
C635: 137 *****
C636: 138 *****
C637: 139 *****
C638: 140 *****
C639: 141 *****
C640: 142 *****
C641: 143 *****
C642: 144 *****
C643: 145 *****
C644: 146 *****
C645: 147 *****
C646: 148 *****
C647: 149 *****
C648: 150 *****
C649: 151 *****
C650: 152 *****
C651: 153 *****
C652: 154 *****
C653: 155 *****
C654: 156 *****
C655: 157 *****
C656: 158 *****
C657: 159 *****
C658: 160 *****
C659: 161 *****
C660: 162 *****
C661: 163 *****
C662: 164 *****
C663: 165 *****
C664: 166 *****
C665: 167 *****
C666: 168 *****
C667: 169 *****
C668: 170 *****
C669: 171 *****
C670: 172 *****
C671: 173 *****
C672: 174 *****
C673: 175 *****
C674: 176 *****
C675: 177 *****
C676: 178 *****
C677: 179 *****
C678: 180 *****
C679: 181 *****
C680: 182 *****
C681: 183 *****
C682: 184 *****
C683: 185 *****
C684: 186 *****
C685: 187 *****
C686: 188 *****
C687: 189 *****
C688: 190 *****
C689: 191 *****
C690: 192 *****
C691: 193 *****
C692: 194 *****
C693: 195 *****
C694: 196 *****
C695: 197 *****
C696: 198 *****
C697: 199 *****
C698: 200 *****
C699: 201 *****
C700: 202 *****
C701: 203 *****
C702: 204 *****
C703: 205 *****
C704: 206 *****
C705: 207 *****
C706: 208 *****
C707: 209 *****
C708: 210 *****
C709: 211 *****
C710: 212 *****
C711: 213 *****
C712: 214 *****
C713: 215 *****
C714: 216 *****
C715: 217 *****
C716: 218 *****
C717: 219 *****
C718: 220 *****
C719: 221 *****
C720: 222 *****
C721: 223 *****
C722: 224 *****
C723: 225 *****
C724: 226 *****
C725: 227 *****
C726: 228 *****
C727: 229 *****
C728: 230 *****
C729: 231 *****
C730: 232 *****
C731: 233 *****
C732: 234 *****
C733: 235 *****
C734: 236 *****
C735: 237 *****
C736: 238 *****
C737: 239 *****
C738: 240 *****
C739: 241 *****
C740: 242 *****
C741: 243 *****
C742: 244 *****
C743: 245 *****
C744: 246 *****
C745: 247 *****
C746: 248 *****
C747: 249 *****
C748: 250 *****
C749: 251 *****
C750: 252 *****
C751: 253 *****
C752: 254 *****
C753: 255 *****
C754: 256 *****
C755: 257 *****
C756: 258 *****
C757: 259 *****
C758: 260 *****
C759: 261 *****
C760: 262 *****
C761: 263 *****
C762: 264 *****
C763: 265 *****
C764: 266 *****
C765: 267 *****
C766: 268 *****
C767: 269 *****
C768: 270 *****
C769: 271 *****
C770: 272 *****
C771: 273 *****
C772: 274 *****
C773: 275 *****
C774: 276 *****
C775: 277 *****
C776: 278 *****
C777: 279 *****
C778: 280 *****
C779: 281 *****
C780: 282 *****
C781: 283 *****
C782: 284 *****
C783: 285 *****
C784: 286 *****
C785: 287 *****
C786: 288 *****
C787: 289 *****
C788: 290 *****
C789: 291 *****
C790: 292 *****
C791: 293 *****
C792: 294 *****
C793: 295 *****
C794: 296 *****
C795: 297 *****
C796: 298 *****
C797: 299 *****
C798: 300 *****
C799: 301 *****
C800: 302 *****
C801: 303 *****
C802: 304 *****
C803: 305 *****
C804: 306 *****
C805: 307 *****
C806: 308 *****
C807: 309 *****
C808: 310 *****
C809: 311 *****
C810: 312 *****
C811: 313 *****
C812: 314 *****
C813: 315 *****
C814: 316 *****
C815: 317 *****
C816: 318 *****
C817: 319 *****
C818: 320 *****
C819: 321 *****
C820: 322 *****
C821: 323 *****
C822: 324 *****
C823: 325 *****
C824: 326 *****
C825: 327 *****
C826: 328 *****
C827: 329 *****
C828: 330 *****
C829: 331 *****
C830: 332 *****
C831: 333 *****
C832: 334 *****
C833: 335 *****
C834: 336 *****
C835: 337 *****
C836: 338 *****
C837: 339 *****
C838: 340 *****
C839: 341 *****
C840: 342 *****
C841: 343 *****
C842: 344 *****
C843: 345 *****
C844: 346 *****
C845: 347 *****
C846: 348 *****
C847: 349 *****
C848: 350 *****
C849: 351 *****
C850: 352 *****
C851: 353 *****
C852: 354 *****
C853: 355 *****
C854: 356 *****
C855: 357 *****
C856: 358 *****
C857: 359 *****
C858: 360 *****
C859: 361 *****
C860: 362 *****
C861: 363 *****
C862: 364 *****
C863: 365 *****
C864: 366 *****
C865: 367 *****
C866: 368 *****
C867: 369 *****
C868: 370 *****
C869: 371 *****
C870: 372 *****
C871: 373 *****
C872: 374 *****
C873: 375 *****
C874: 376 *****
C875: 377 *****
C876: 378 *****
C877: 379 *****
C878: 380 *****
C879: 381 *****
C880: 382 *****
C881: 383 *****
C882: 384 *****
C883: 385 *****
C884: 386 *****
C885: 387 *****
C886: 388 *****
C887: 389 *****
C888: 390 *****
C889: 391 *****
C890: 392 *****
C891: 393 *****
C892: 394 *****
C893: 395 *****
C894: 396 *****
C895: 397 *****
C896: 398 *****
C897: 399 *****
C898: 400 *****
C899: 401 *****
C900: 402 *****
C901: 403 *****
C902: 404 *****
C903: 405 *****
C904: 406 *****
C905: 407 *****
C906: 408 *****
C907: 409 *****
C908: 410 *****
C909: 411 *****
C910: 412 *****
C911: 413 *****
C912: 414 *****
C913: 415 *****
C914: 416 *****
C915: 417 *****
C916: 418 *****
C917: 419 *****
C918: 420 *****
C919: 421 *****
C920: 422 *****
C921: 423 *****
C922: 424 *****
C923: 425 *****
C924: 426 *****
C925: 427 *****
C926: 428 *****
C927: 429 *****
C928: 430 *****
C929: 431 *****
C930: 432 *****
C931: 433 *****
C932: 434 *****
C933: 435 *****
C934: 436 *****
C935: 437 *****
C936: 438 *****
C937: 439 *****
C938: 440 *****
C939: 441 *****
C940: 442 *****
C941: 443 *****
C942: 444 *****
C943: 445 *****
C944: 446 *****
C945: 447 *****
C946: 448 *****
C947: 449 *****
C948: 450 *****
C949: 451 *****
C950: 452 *****
C951: 453 *****
C952: 454 *****
C953: 455 *****
C954: 456 *****
C955: 457 *****
C956: 458 *****
C957: 459 *****
C958: 460 *****
C959: 461 *****
C960: 462 *****
C961: 463 *****
C962: 464 *****
C963: 465 *****
C964: 466 *****
C965: 467 *****
C966: 468 *****
C967: 469 *****
C968: 470 *****
C969: 471 *****
C970: 472 *****
C971: 473 *****
C972: 474 *****
C973: 475 *****
C974: 476 *****
C975: 477 *****
C976: 478 *****
C977: 479 *****
C978: 480 *****
C979: 481 *****
C980: 482 *****
C981: 483 *****
C982: 484 *****
C983: 485 *****
C984: 486 *****
C985: 487 *****
C986: 488 *****
C987: 489 *****
C988: 490 *****
C989: 491 *****
C990: 492 *****
C991: 493 *****
C992: 494 *****
C993: 495 *****
C994: 496 *****
C995: 497 *****
C996: 498 *****
C997: 499 *****
C998: 500 *****
C999: 501 *****

```

```

C5C:49 A5 70 EOR #A5
C5D:8D F4 03 STA SOFTEV*2
C5D:90 EA CSBF BRA GETCODE3
;
C5D5: C5D5 74 GETCODES EQU *
C5D6: C5D6 75 PTRBUFF*1 INC PTRBUFF*1
C5D7: C5D7 76 INC PTRBUFF*1
C5D8: C5D8 77 BRA GETCODE2
;

```

```

C5DB: C5DB 81 ARAPPLETALK2 EQU *
C5DB:80 PHP
C5D:08 CLD
C5D:09 CLD
C5D:0A STY
C5D:0C 78 04 STY SCRNTMP0
;
C5E1:A0 0B LDY #2P2CUSELEN
C5E3:09 BF C5E3 EQU #2P2CUSELEN
C5E4:09 BF 00 LDY #2P2CUSE-1,Y
C5E7:A0 PHA
C5E7:08 DEY
C5E8:D0 F9 BNE ARSTKSVE
;

```

```

;Reg AppleTalk call conta here
;Make sure no ints in here
;MUST enter with Dec mode clear
;Save Y temporarily
;Save Y temporarily

```

```

C5F1:A0 0B LDY #2P2CUSELEN
C5F3:09 BF C5F3 EQU #2P2CUSE-1,Y
C5F4:09 BF 00 LDY #2P2CUSE-1,Y
C5F7:A0 PHA
C5F7:08 DEY
C5F8:D0 F9 BNE ARSTKSVE
;
C5F9:0C C9 STX ADDR0
C5F9:0E 78 04 LDX SCRNTMP#
C5F9:0F CA STX SCRNTMP#
C5F1:B1 C9 LDA (ADDR0),Y
C5F3:F0 7A BEQ ARAPTALK2
C5F5:30 78 BNE ARAPTALK2
C5F7:08 18 BCS CMDEXITE
C5F8:0A 78 ASL
C5F8:0A 18 BCS CMDEXITE
C5F9:0A 18 TAX
C5FD:C8 INY
C5FE:B1 C9 LDA (ADDR0),Y
C5FF:0C 74 C7 INY
C5FF:0E 74 C7 INY
C5FF:10 74 C7 JMP (APTALKCMDS-2,X) ; jump to routine

```

```

;0 of bytes to save on stk
;Get value to save
;Save it
;Test for more
;If >= go for more
;Use data buffer ptr
;Dec it by 1
;Hi byte of data buff ptr
;Get command
;0 is invalid command
;- then test for DIAG call
;save test for valid
;Get test for valid
;Make command into index
;
;Inc to 2nd byte in user buff
;Pick up the data there
;Inc index for later
;Jump to routine

```

44	APTKAL_ROMSTUFF	AppleTalk init entry point	29-JUL-85	19:33	PAGE 11
C684	110	ARINIT	EQ	*ApplTalk init cell entry	
C684	111	STZ	TESTTMP	*=0 indicates from here	
C686	114	ARINIT1	EQ	*ApplTalk init cell entry	
C686	115	ARINIT2	EQ	*APTKALKUNIT	
C689	116	INC	APTKALKUNIT	*Save in screen hole	
C689	117	LDA	#CMDCID1	*Commands code #	
C689	118	STZ	CALLSETUP	*Get up screen stuff	
C689	119	JSR	CALLSETUP	*Get up screen stuff	
C689	120	STA	PIRBUFF+1	*some device sends us	
C689	121	STZ	PIRBUFF	*Lo byte is zero	
C689	122	STZ			
C689	123	JSR	ALTRPCNVENTORY	*Call the protocol converter	
C689	124	DFB	POSTATUSCMD	*prot Conv status command	
C689	125	DMF	ZPCUSE	*Pointer to call buffer ???	
C689	126	DMF	PIRBUFF	*Status in PIRBUFF	
C689	127	BNE	NOTHISUNIT	*Not ID1 then maybe last unit	
C689	128	INC	CMDCMDS	*Commands code now CMDCID2	
C689	129	JSR	ALTRPCNVENTORY	*Call the protocol converter	
C689	130	DFB	ZSTATUSCMD	*prot Conv status command	
C689	131	DMF	ZPCUSE	*Pointer to call buffer	
C689	132	DMF	PIRBUFF	*Status in PIRBUFF	
C689	133	CMF	#ID2-888	*ApplTalk ID status code 2??	
C689	134	BNE	NOTHISUNIT	*Not ID2 then maybe last unit	
C689	135	CLC		*If here then we've got it	
C689	136	BRA	MAYCONTINIT	*See if we should cont INIT call	
C689	137	CMF	#NODEVCN-888	*Test for dev not connected	
C689	138	BNE	ARINIT4	*C=1 if bad unit =1 tried	
C689	139	LDA	TESTTMP	*unit call=<888>reboot cell	
C689	140	STZ	ARINIT4	*Reset unit in init	
C689	141	RTS		*V=1 return to reboot call	
C689	142	INC	ARINIT4	*Continue unit call here	
C689	143	EQ	DOEXIT1	*C=1 then ApplTalk unit not avail	
C689	144	PCS	DOEXIT1	*Commands code for init	
C689	145	BRA	ARINIT4	*Skip ARSTATUS entry point	
C689	146	ARINIT4	EQ		
C689	147	PCS	DOEXIT1		
C689	148	BRA	ARINIT4		
C689	149	BRA	ARINIT4		
C689	150	ARSTATUS	EQ	*Alt ROM status entry point	
C689	151	LDY	#CMDCSTATUS	*Command code for status	
C689	152	INC	ARSTATUS		
C689	153	INC	ARSTATUS		
C689	154	ARINIT6	EQ	*Calculate user buffer	
C689	155	INC	STUPTPRS	*C=8 then leave hi byte alone	
C689	156	INC	STUPTPRS		
C689	157	INC	ADRT1		
C689	158	STUPTPRS	LDA	(DOERR)	
C689	159	STUPTPRS	LDA	BYTEUSER	
C689	160	STA	ADRT1	*Put in buffer	
C689	161	LDA	ADRT1	*Put in buffer	
C689	162	STA	ADRT1		
C689	163	LDA	ADRT1		
C689	164	STA	ADRT1		
C689	165	TYA	PIRBUFF+1	*Move commands code to 'A'	
C689	166	CALLBOX	EQ		
C689	167	CALLBOX	EQ		
C689	168	JSR	CALLSETUP	*Setup some stuff for JSR	
C689	169	DFB	POSTATUSCMD	*prot Conv status command	
C689	170	DFB	ZPCUSE	*Pointer to call buffer	
C689	171	DMF	ZPCUSE		

```

29-JUL-05 10:33 PAGE 12

94 APTAKL.ROMSTUFF Exit ApTakt cmd rtns

C65D 174 DOEXIT EQU
C65E 175 BEQ
C65F 176 LDR #0
C660 177 BNE
C661 178 DOEXIT EQU
C662 179 DOEXIT1 LDA
C663 180 DOEXIT2 EQU
C664 181 DOEXIT3 EQU
C665 182 TAX
C666 183 BNE
C667 184 NECDXIT EQU
C668 185 LDX
C669 186 CLC
C66A 187 BNA
C66B 188 ARAPTALK2 EQU
C66C 189 CMP
C66D 190 BNE
C66E 191 BEQ
C66F 192 CMDEXITE EQU
C670 193 FF
C671 194 FF
C672 195 LDX
C673 196 CMDEXIT EQU
C674 197 SEC
C675 198 CMDEXIT EQU
C676 199 LDX
C677 200 ARSTKRST EQU
C678 201 PLA
C679 202 STA
C67A 203 BNE
C67B 204 BNE
C67C 205 CLV
C67D 206 AND
C67E 207 AND
C67F 208 BNE
C680 209
C681 210
C682 211 NOTACTIVE
C683 212 NOTACTIVE
C684 213 JMP
C685 214 JMP
C686 215 JMP
C687 216 JMP
C688 217 JMP
C689 218 JMP
C68A 219 JMP
C68B 220 JMP
C68C 221 JMP
C68D 222 JMP
C68E 223 JMP
C68F 224 JMP
C690 225 JMP
C691 226 JMP
C692 227 JMP
C693 228 JMP
C694 229 JMP
C695 230 JMP
C696 231 JMP
C697 232 JMP
C698 233 JMP
C699 234 JMP
C69A 235 JMP
C69B 236 JMP
C69C 237 JMP
C69D 238 JMP
C69E 239 JMP
C69F 240 JMP
C6A0 241 JMP
C6A1 242 JMP
C6A2 243 JMP
C6A3 244 JMP
C6A4 245 JMP
C6A5 246 JMP
C6A6 247 JMP
C6A7 248 JMP
C6A8 249 JMP
C6A9 250 JMP
C6AA 251 JMP
C6AB 252 JMP
C6AC 253 JMP
C6AD 254 JMP
C6AE 255 JMP
C6AF 256 JMP
C6B0 257 JMP
C6B1 258 JMP
C6B2 259 JMP
C6B3 260 JMP
C6B4 261 JMP
C6B5 262 JMP
C6B6 263 JMP
C6B7 264 JMP
C6B8 265 JMP
C6B9 266 JMP
C6BA 267 JMP
C6BB 268 JMP
C6BC 269 JMP
C6BD 270 JMP
C6BE 271 JMP
C6BF 272 JMP
C6C0 273 JMP
C6C1 274 JMP
C6C2 275 JMP
C6C3 276 JMP
C6C4 277 JMP
C6C5 278 JMP
C6C6 279 JMP
C6C7 280 JMP
C6C8 281 JMP
C6C9 282 JMP
C6CA 283 JMP
C6CB 284 JMP
C6CC 285 JMP
C6CD 286 JMP
C6CE 287 JMP
C6CF 288 JMP
C6D0 289 JMP
C6D1 290 JMP
C6D2 291 JMP
C6D3 292 JMP
C6D4 293 JMP
C6D5 294 JMP
C6D6 295 JMP
C6D7 296 JMP
C6D8 297 JMP
C6D9 298 JMP
C6DA 299 JMP
C6DB 300 JMP
C6DC 301 JMP
C6DD 302 JMP
C6DE 303 JMP
C6DF 304 JMP
C6E0 305 JMP
C6E1 306 JMP
C6E2 307 JMP
C6E3 308 JMP
C6E4 309 JMP
C6E5 310 JMP
C6E6 311 JMP
C6E7 312 JMP
C6E8 313 JMP
C6E9 314 JMP
C6EA 315 JMP
C6EB 316 JMP
C6EC 317 JMP
C6ED 318 JMP
C6EE 319 JMP
C6EF 320 JMP
C6F0 321 JMP
C6F1 322 JMP
C6F2 323 JMP
C6F3 324 JMP
C6F4 325 JMP
C6F5 326 JMP
C6F6 327 JMP
C6F7 328 JMP
C6F8 329 JMP
C6F9 330 JMP
C6FA 331 JMP
C6FB 332 JMP
C6FC 333 JMP
C6FD 334 JMP
C6FE 335 JMP
C6FF 336 JMP
C700 337 JMP
C701 338 JMP
C702 339 JMP
C703 340 JMP
C704 341 JMP
C705 342 JMP
C706 343 JMP
C707 344 JMP
C708 345 JMP
C709 346 JMP
C70A 347 JMP
C70B 348 JMP
C70C 349 JMP
C70D 350 JMP
C70E 351 JMP
C70F 352 JMP
C710 353 JMP
C711 354 JMP
C712 355 JMP
C713 356 JMP
C714 357 JMP
C715 358 JMP
C716 359 JMP
C717 360 JMP
C718 361 JMP
C719 362 JMP
C71A 363 JMP
C71B 364 JMP
C71C 365 JMP
C71D 366 JMP
C71E 367 JMP
C71F 368 JMP
C720 369 JMP
C721 370 JMP
C722 371 JMP
C723 372 JMP
C724 373 JMP
C725 374 JMP
C726 375 JMP
C727 376 JMP
C728 377 JMP
C729 378 JMP
C72A 379 JMP
C72B 380 JMP
C72C 381 JMP
C72D 382 JMP
C72E 383 JMP
C72F 384 JMP
C730 385 JMP
C731 386 JMP
C732 387 JMP
C733 388 JMP
C734 389 JMP
C735 390 JMP
C736 391 JMP
C737 392 JMP
C738 393 JMP
C739 394 JMP
C73A 395 JMP
C73B 396 JMP
C73C 397 JMP
C73D 398 JMP
C73E 399 JMP
C73F 400 JMP
C740 401 JMP
C741 402 JMP
C742 403 JMP
C743 404 JMP
C744 405 JMP
C745 406 JMP
C746 407 JMP
C747 408 JMP
C748 409 JMP
C749 410 JMP
C74A 411 JMP
C74B 412 JMP
C74C 413 JMP
C74D 414 JMP
C74E 415 JMP
C74F 416 JMP
C750 417 JMP
C751 418 JMP
C752 419 JMP
C753 420 JMP
C754 421 JMP
C755 422 JMP
C756 423 JMP
C757 424 JMP
C758 425 JMP
C759 426 JMP
C75A 427 JMP
C75B 428 JMP
C75C 429 JMP
C75D 430 JMP
C75E 431 JMP
C75F 432 JMP
C760 433 JMP
C761 434 JMP
C762 435 JMP
C763 436 JMP
C764 437 JMP
C765 438 JMP
C766 439 JMP
C767 440 JMP
C768 441 JMP
C769 442 JMP
C76A 443 JMP
C76B 444 JMP
C76C 445 JMP
C76D 446 JMP
C76E 447 JMP
C76F 448 JMP
C770 449 JMP
C771 450 JMP
C772 451 JMP
C773 452 JMP
C774 453 JMP
C775 454 JMP
C776 455 JMP
C777 456 JMP
C778 457 JMP
C779 458 JMP
C77A 459 JMP
C77B 460 JMP
C77C 461 JMP
C77D 462 JMP
C77E 463 JMP
C77F 464 JMP
C780 465 JMP
C781 466 JMP
C782 467 JMP
C783 468 JMP
C784 469 JMP
C785 470 JMP
C786 471 JMP
C787 472 JMP
C788 473 JMP
C789 474 JMP
C78A 475 JMP
C78B 476 JMP
C78C 477 JMP
C78D 478 JMP
C78E 479 JMP
C78F 480 JMP
C790 481 JMP
C791 482 JMP
C792 483 JMP
C793 484 JMP
C794 485 JMP
C795 486 JMP
C796 487 JMP
C797 488 JMP
C798 489 JMP
C799 490 JMP
C79A 491 JMP
C79B 492 JMP
C79C 493 JMP
C79D 494 JMP
C79E 495 JMP
C79F 496 JMP
C800 497 JMP
C801 498 JMP
C802 499 JMP
C803 500 JMP
C804 501 JMP
C805 502 JMP
C806 503 JMP
C807 504 JMP
C808 505 JMP
C809 506 JMP
C80A 507 JMP
C80B 508 JMP
C80C 509 JMP
C80D 510 JMP
C80E 511 JMP
C80F 512 JMP
C810 513 JMP
C811 514 JMP
C812 515 JMP
C813 516 JMP
C814 517 JMP
C815 518 JMP
C816 519 JMP
C817 520 JMP
C818 521 JMP
C819 522 JMP
C81A 523 JMP
C81B 524 JMP
C81C 525 JMP
C81D 526 JMP
C81E 527 JMP
C81F 528 JMP
C820 529 JMP
C821 530 JMP
C822 531 JMP
C823 532 JMP
C824 533 JMP
C825 534 JMP
C826 535 JMP
C827 536 JMP
C828 537 JMP
C829 538 JMP
C82A 539 JMP
C82B 540 JMP
C82C 541 JMP
C82D 542 JMP
C82E 543 JMP
C82F 544 JMP
C830 545 JMP
C831 546 JMP
C832 547 JMP
C833 548 JMP
C834 549 JMP
C835 550 JMP
C836 551 JMP
C837 552 JMP
C838 553 JMP
C839 554 JMP
C83A 555 JMP
C83B 556 JMP
C83C 557 JMP
C83D 558 JMP
C83E 559 JMP
C83F 560 JMP
C840 561 JMP
C841 562 JMP
C842 563 JMP
C843 564 JMP
C844 565 JMP
C845 566 JMP
C846 567 JMP
C847 568 JMP
C848 569 JMP
C849 570 JMP
C84A 571 JMP
C84B 572 JMP
C84C 573 JMP
C84D 574 JMP
C84E 575 JMP
C84F 576 JMP
C850 577 JMP

```



```

C6F4: 293 * Now move data to write into card. The
C6F4: 294 * data is obtained from the table pointed
C6F4: 295 * to in the WRITE parameter list.
C6F4: 296 * The WRITE parameter list is set up as
C6F4: 297 * follows:
C6F4: 298 *
C6F4: 299 * WRITETBL EQU *
C6F4: 300 * DW 1
C6F4: 301 * DW addr of dest addr
C6F4: 302 * DW 1
C6F4: 303 * DW addr of src addr
C6F4: 304 * DW 1
C6F4: 305 * DW addr of LAP type
C6F4: 306 * DW bbbb
C6F4: 307 * DW addr of DDP data
C6F4: 308 * DW bbbb
C6F4: 309 * DW addr of ATP data
C6F4: 310 * DW bbbb
C6F4: 311 * DW addr of misc data
C6F4: 312 * DW $FFxx
C6F4: 313 * Terminator - REQUIRED
C6F4: 314
C6F4: 315 ARRWITE EQU *
C6F4: 316 TRX
C6F4: 317 STA (ADDR0),Y
C6F4: 318 STA ADDR0
C6F4: 319 TRX
C6F4: 320 TESTTMP
C6F4: 321 STZ TESTTMP+1
C6F4: 322 STZ
C6F4: 323 SEND2MANYLP EQU *
C6F4: 324 LDA (ADDR0),Y
C6F4: 325 ADC TESTTMP
C6F4: 326 STA
C6F4: 327 INY
C6F4: 328 STA
C6F4: 329 INC
C6F4: 330 FOUNDEND
C6F4: 331 DEC
C6F4: 332 ADC
C6F4: 333 STZ
C6F4: 334 INY
C6F4: 335 INY
C6F4: 336 INY
C6F4: 337 BNE
C6F4: 338 INC
C6F4: 339 BRA SEND2MANYLP
C6F4: 340
C6F4: 341 FOUNDEND EQU *
C6F4: 342 STX ADDR1
C6F4: 343 LDA TESTTMP+1
C6F4: 344 CMP #3
C6F4: 345 STC SHORTENUF
C6F4: 346 LDX #BYTEGR685-68
C6F4: 347 JNP ECNEXIT
C6F4: 348
C6F4: 349 ITSHORTENUF EQU *
C6F4: 350 LDY
C6F4: 351 LDA
C6F4: 352 CALLSETUP2
C6F4: 353 STZ TYPEWRITE
C6F4: 354
C6F4: 355 ARRWITE2 EQU *
C6F4: 356 LDA (ADDR0),Y
C6F4: 357 STA ADDR1
C6F4: 358 TRX
C6F4: 359 LDA (ADDR0),Y
C6F4: 360 INY
C6F4: 361 CMP #FF
C6F4: 362 BEQ SAYSENDIT
C6F4: 363 STA NUMHIWRITE
C6F4: 364 INY
C6F4: 365
C6F4: 366
C6F4: 367
C6F4: 368
C6F4: 369
C6F4: 370
C6F4: 371
C6F4: 372
C6F4: 373
C6F4: 374
C6F4: 375
C6F4: 376
C6F4: 377
C6F4: 378
C6F4: 379
C6F4: 380
C6F4: 381
C6F4: 382
C6F4: 383
C6F4: 384
C6F4: 385
C6F4: 386
C6F4: 387
C6F4: 388
C6F4: 389
C6F4: 390
C6F4: 391
C6F4: 392
C6F4: 393
C6F4: 394
C6F4: 395
C6F4: 396
C6F4: 397
C6F4: 398
C6F4: 399
C6F4: 400
C6F4: 401
C6F4: 402
C6F4: 403
C6F4: 404
C6F4: 405
C6F4: 406
C6F4: 407
C6F4: 408
C6F4: 409
C6F4: 410
C6F4: 411
C6F4: 412
C6F4: 413
C6F4: 414
C6F4: 415
C6F4: 416
C6F4: 417
C6F4: 418
C6F4: 419
C6F4: 420
C6F4: 421
C6F4: 422
C6F4: 423
C6F4: 424
C6F4: 425
C6F4: 426
C6F4: 427
C6F4: 428
C6F4: 429
C6F4: 430
C6F4: 431
C6F4: 432
C6F4: 433
C6F4: 434
C6F4: 435
C6F4: 436
C6F4: 437
C6F4: 438
C6F4: 439
C6F4: 440
C6F4: 441
C6F4: 442
C6F4: 443
C6F4: 444
C6F4: 445
C6F4: 446
C6F4: 447
C6F4: 448
C6F4: 449
C6F4: 450
C6F4: 451
C6F4: 452
C6F4: 453
C6F4: 454
C6F4: 455
C6F4: 456
C6F4: 457
C6F4: 458
C6F4: 459
C6F4: 460
C6F4: 461
C6F4: 462
C6F4: 463
C6F4: 464
C6F4: 465
C6F4: 466
C6F4: 467
C6F4: 468
C6F4: 469
C6F4: 470
C6F4: 471
C6F4: 472
C6F4: 473
C6F4: 474
C6F4: 475
C6F4: 476
C6F4: 477
C6F4: 478
C6F4: 479
C6F4: 480
C6F4: 481
C6F4: 482
C6F4: 483
C6F4: 484
C6F4: 485
C6F4: 486
C6F4: 487
C6F4: 488
C6F4: 489
C6F4: 490
C6F4: 491
C6F4: 492
C6F4: 493
C6F4: 494
C6F4: 495
C6F4: 496
C6F4: 497
C6F4: 498
C6F4: 499
C6F4: 500
C6F4: 501
C6F4: 502
C6F4: 503
C6F4: 504
C6F4: 505
C6F4: 506
C6F4: 507
C6F4: 508
C6F4: 509
C6F4: 510
C6F4: 511
C6F4: 512
C6F4: 513
C6F4: 514
C6F4: 515
C6F4: 516
C6F4: 517
C6F4: 518
C6F4: 519
C6F4: 520
C6F4: 521
C6F4: 522
C6F4: 523
C6F4: 524
C6F4: 525
C6F4: 526
C6F4: 527
C6F4: 528
C6F4: 529
C6F4: 530
C6F4: 531
C6F4: 532
C6F4: 533
C6F4: 534
C6F4: 535
C6F4: 536
C6F4: 537
C6F4: 538
C6F4: 539
C6F4: 540
C6F4: 541
C6F4: 542
C6F4: 543
C6F4: 544
C6F4: 545
C6F4: 546
C6F4: 547
C6F4: 548
C6F4: 549
C6F4: 550
C6F4: 551
C6F4: 552
C6F4: 553
C6F4: 554
C6F4: 555
C6F4: 556
C6F4: 557
C6F4: 558
C6F4: 559
C6F4: 560
C6F4: 561
C6F4: 562
C6F4: 563
C6F4: 564
C6F4: 565
C6F4: 566
C6F4: 567
C6F4: 568
C6F4: 569
C6F4: 570
C6F4: 571
C6F4: 572
C6F4: 573
C6F4: 574
C6F4: 575
C6F4: 576
C6F4: 577
C6F4: 578
C6F4: 579
C6F4: 580
C6F4: 581
C6F4: 582
C6F4: 583
C6F4: 584
C6F4: 585
C6F4: 586
C6F4: 587
C6F4: 588
C6F4: 589
C6F4: 590
C6F4: 591
C6F4: 592
C6F4: 593
C6F4: 594
C6F4: 595
C6F4: 596
C6F4: 597
C6F4: 598
C6F4: 599
C6F4: 600
C6F4: 601
C6F4: 602
C6F4: 603
C6F4: 604
C6F4: 605
C6F4: 606
C6F4: 607
C6F4: 608
C6F4: 609
C6F4: 610
C6F4: 611
C6F4: 612
C6F4: 613
C6F4: 614
C6F4: 615
C6F4: 616
C6F4: 617
C6F4: 618
C6F4: 619
C6F4: 620
C6F4: 621
C6F4: 622
C6F4: 623
C6F4: 624
C6F4: 625
C6F4: 626
C6F4: 627
C6F4: 628
C6F4: 629
C6F4: 630
C6F4: 631
C6F4: 632
C6F4: 633
C6F4: 634
C6F4: 635
C6F4: 636
C6F4: 637
C6F4: 638
C6F4: 639
C6F4: 640
C6F4: 641
C6F4: 642
C6F4: 643
C6F4: 644
C6F4: 645
C6F4: 646
C6F4: 647
C6F4: 648
C6F4: 649
C6F4: 650
C6F4: 651
C6F4: 652
C6F4: 653
C6F4: 654
C6F4: 655
C6F4: 656
C6F4: 657
C6F4: 658
C6F4: 659
C6F4: 660
C6F4: 661
C6F4: 662
C6F4: 663
C6F4: 664
C6F4: 665
C6F4: 666
C6F4: 667
C6F4: 668
C6F4: 669
C6F4: 670
C6F4: 671
C6F4: 672
C6F4: 673
C6F4: 674
C6F4: 675
C6F4: 676
C6F4: 677
C6F4: 678
C6F4: 679
C6F4: 680
C6F4: 681
C6F4: 682
C6F4: 683
C6F4: 684
C6F4: 685
C6F4: 686
C6F4: 687
C6F4: 688
C6F4: 689
C6F4: 690
C6F4: 691
C6F4: 692
C6F4: 693
C6F4: 694
C6F4: 695
C6F4: 696
C6F4: 697
C6F4: 698
C6F4: 699
C6F4: 700
C6F4: 701
C6F4: 702
C6F4: 703
C6F4: 704
C6F4: 705
C6F4: 706
C6F4: 707
C6F4: 708
C6F4: 709
C6F4: 710
C6F4: 711
C6F4: 712
C6F4: 713
C6F4: 714
C6F4: 715
C6F4: 716
C6F4: 717
C6F4: 718
C6F4: 719
C6F4: 720
C6F4: 721
C6F4: 722
C6F4: 723
C6F4: 724
C6F4: 725
C6F4: 726
C6F4: 727
C6F4: 728
C6F4: 729
C6F4: 730
C6F4: 731
C6F4: 732
C6F4: 733
C6F4: 734
C6F4: 735
C6F4: 736
C6F4: 737
C6F4: 738
C6F4: 739
C6F4: 740
C6F4: 741
C6F4: 742
C6F4: 743
C6F4: 744
C6F4: 745
C6F4: 746
C6F4: 747
C6F4: 748
C6F4: 749
C6F4: 750
C6F4: 751
C6F4: 752
C6F4: 753
C6F4: 754
C6F4: 755
C6F4: 756
C6F4: 757
C6F4: 758
C6F4: 759
C6F4: 760
C6F4: 761
C6F4: 762
C6F4: 763
C6F4: 764
C6F4: 765
C6F4: 766
C6F4: 767
C6F4: 768
C6F4: 769
C6F4: 770
C6F4: 771
C6F4: 772
C6F4: 773
C6F4: 774
C6F4: 775
C6F4: 776
C6F4: 777
C6F4: 778
C6F4: 779
C6F4: 780
C6F4: 781
C6F4: 782
C6F4: 783
C6F4: 784
C6F4: 785
C6F4: 786
C6F4: 787
C6F4: 788
C6F4: 789
C6F4: 790
C6F4: 791
C6F4: 792
C6F4: 793
C6F4: 794
C6F4: 795
C6F4: 796
C6F4: 797
C6F4: 798
C6F4: 799
C6F4: 800
C6F4: 801
C6F4: 802
C6F4: 803
C6F4: 804
C6F4: 805
C6F4: 806
C6F4: 807
C6F4: 808
C6F4: 809
C6F4: 810
C6F4: 811
C6F4: 812
C6F4: 813
C6F4: 814
C6F4: 815
C6F4: 816
C6F4: 817
C6F4: 818
C6F4: 819
C6F4: 820
C6F4: 821
C6F4: 822
C6F4: 823
C6F4: 824
C6F4: 825
C6F4: 826
C6F4: 827
C6F4: 828
C6F4: 829
C6F4: 830
C6F4: 831
C6F4: 832
C6F4: 833
C6F4: 834
C6F4: 835
C6F4: 836
C6F4: 837
C6F4: 838
C6F4: 839
C6F4: 840
C6F4: 841
C6F4: 842
C6F4: 843
C6F4: 844
C6F4: 845
C6F4: 846
C6F4: 847
C6F4: 848
C6F4: 849
C6F4: 850
C6F4: 851
C6F4: 852
C6F4: 853
C6F4: 854
C6F4: 855
C6F4: 856
C6F4: 857
C6F4: 858
C6F4: 859
C6F4: 860
C6F4: 861
C6F4: 862
C6F4: 863
C6F4: 864
C6F4: 865
C6F4: 866
C6F4: 867
C6F4: 868
C6F4: 869
C6F4: 870
C6F4: 871
C6F4: 872
C6F4: 873
C6F4: 874
C6F4: 875
C6F4: 876
C6F4: 877
C6F4: 878
C6F4: 879
C6F4: 880
C6F4: 881
C6F4: 882
C6F4: 883
C6F4: 884
C6F4: 885
C6F4: 886
C6F4: 887
C6F4: 888
C6F4: 889
C6F4: 890
C6F4: 891
C6F4: 892
C6F4: 893
C6F4: 894
C6F4: 895
C6F4: 896
C6F4: 897
C6F4: 898
C6F4: 899
C6F4: 900
C6F4: 901
C6F4: 902
C6F4: 903
C6F4: 904
C6F4: 905
C6F4: 906
C6F4: 907
C6F4: 908
C6F4: 909
C6F4: 910
C6F4: 911
C6F4: 912
C6F4: 913
C6F4: 914
C6F4: 915
C6F4: 916
C6F4: 917
C6F4: 918
C6F4: 919
C6F4: 920
C6F4: 921
C6F4: 922
C6F4: 923
C6F4: 924
C6F4: 925
C6F4: 926
C6F4: 927
C6F4: 928
C6F4: 929
C6F4: 930
C6F4: 931
C6F4: 932
C6F4: 933
C6F4: 934
C6F4: 935
C6F4: 936
C6F4: 937
C6F4: 938
C6F4: 939
C6F4: 940
C6F4: 941
C6F4: 942
C6F4: 943
C6F4: 944
C6F4: 945
C6F4: 946
C6F4: 947
C6F4: 948
C6F4: 949
C6F4: 950
C6F4: 951
C6F4: 952
C6F4: 953
C6F4: 954
C6F4: 955
C6F4: 956
C6F4: 957
C6F4: 958
C6F4: 959
C6F4: 960
C6F4: 961
C6F4: 962
C6F4: 963
C6F4: 964
C6F4: 965
C6F4: 966
C6F4: 967
C6F4: 968
C6F4: 969
C6F4: 970
C6F4: 971
C6F4: 972
C6F4: 973
C6F4: 974
C6F4: 975
C6F4: 976
C6F4: 977
C6F4: 978
C6F4: 979
C6F4: 980
C6F4: 981
C6F4: 982
C6F4: 983
C6F4: 984
C6F4: 985
C6F4: 986
C6F4: 987
C6F4: 988
C6F4: 989
C6F4: 990
C6F4: 991
C6F4: 992
C6F4: 993
C6F4: 994
C6F4: 995
C6F4: 996
C6F4: 997
C6F4: 998
C6F4: 999
C6F4: 1000

```

```

C740:B1 C9 LDA (ADDR0),Y
C742:85 C2 STA PTRBUFF
C744:1C B1 INY
C745:B1 C9 LDA (ADDR0),Y
C747:1C B1 STA PTRBUFF+1
C749:1C B1 INY
C74A:1C B1 INY
C74B:1C B1 INY
C74C:1C B1 INY
C74D:1C B1 INY
C74E:1C B1 INY
C74F:1C B1 INY
C750:1C B1 INY
C751:1C B1 INY
C752:1C B1 INY
C753:1C B1 INY
C754:1C B1 INY
C755:1C B1 INY
C756:1C B1 INY
C757:1C B1 INY
C758:1C B1 INY
C759:1C B1 INY
C75A:1C B1 INY
C75B:1C B1 INY
C75C:1C B1 INY
C75D:1C B1 INY
C75E:1C B1 INY
C75F:1C B1 INY
C760:1C B1 INY
C761:1C B1 INY
C762:1C B1 INY
C763:1C B1 INY
C764:1C B1 INY
C765:1C B1 INY
C766:1C B1 INY
C767:1C B1 INY
C768:1C B1 INY
C769:1C B1 INY
C76A:1C B1 INY
C76B:1C B1 INY
C76C:1C B1 INY
C76D:1C B1 INY
C76E:1C B1 INY
C76F:1C B1 INY
C770:1C B1 INY
C771:1C B1 INY
C772:1C B1 INY
C773:1C B1 INY
C774:1C B1 INY
C775:1C B1 INY
C776:1C B1 INY
C777:1C B1 INY
C778:1C B1 INY
C779:1C B1 INY
C77A:1C B1 INY
C77B:1C B1 INY
C77C:1C B1 INY
C77D:1C B1 INY
C77E:1C B1 INY
C77F:1C B1 INY
C780:1C B1 INY
C781:1C B1 INY
C782:1C B1 INY
C783:1C B1 INY
C784:1C B1 INY
C785:1C B1 INY
C786:1C B1 INY
C787:1C B1 INY
C788:1C B1 INY
C789:1C B1 INY
C78A:1C B1 INY
C78B:1C B1 INY
C78C:1C B1 INY
C78D:1C B1 INY
C78E:1C B1 INY
C78F:1C B1 INY
C790:1C B1 INY
C791:1C B1 INY
C792:1C B1 INY
C793:1C B1 INY
C794:1C B1 INY
C795:1C B1 INY
C796:1C B1 INY
C797:1C B1 INY
C798:1C B1 INY
C799:1C B1 INY
C79A:1C B1 INY
C79B:1C B1 INY
C79C:1C B1 INY
C79D:1C B1 INY
C79E:1C B1 INY
C79F:1C B1 INY
C7A0:1C B1 INY
C7A1:1C B1 INY
C7A2:1C B1 INY
C7A3:1C B1 INY
C7A4:1C B1 INY
C7A5:1C B1 INY
C7A6:1C B1 INY
C7A7:1C B1 INY
C7A8:1C B1 INY
C7A9:1C B1 INY
C7AA:1C B1 INY
C7AB:1C B1 INY
C7AC:1C B1 INY
C7AD:1C B1 INY
C7AE:1C B1 INY
C7AF:1C B1 INY
C7B0:1C B1 INY
C7B1:1C B1 INY
C7B2:1C B1 INY
C7B3:1C B1 INY
C7B4:1C B1 INY
C7B5:1C B1 INY
C7B6:1C B1 INY
C7B7:1C B1 INY
C7B8:1C B1 INY
C7B9:1C B1 INY
C7BA:1C B1 INY
C7BB:1C B1 INY
C7BC:1C B1 INY
C7BD:1C B1 INY
C7BE:1C B1 INY
C7BF:1C B1 INY
C7C0:1C B1 INY
C7C1:1C B1 INY
C7C2:1C B1 INY
C7C3:1C B1 INY
C7C4:1C B1 INY
C7C5:1C B1 INY
C7C6:1C B1 INY
C7C7:1C B1 INY
C7C8:1C B1 INY
C7C9:1C B1 INY
C7CA:1C B1 INY
C7CB:1C B1 INY
C7CC:1C B1 INY
C7CD:1C B1 INY
C7CE:1C B1 INY
C7CF:1C B1 INY
C7D0:1C B1 INY
C7D1:1C B1 INY
C7D2:1C B1 INY
C7D3:1C B1 INY
C7D4:1C B1 INY
C7D5:1C B1 INY
C7D6:1C B1 INY
C7D7:1C B1 INY
C7D8:1C B1 INY
C7D9:1C B1 INY
C7DA:1C B1 INY
C7DB:1C B1 INY
C7DC:1C B1 INY
C7DD:1C B1 INY
C7DE:1C B1 INY
C7DF:1C B1 INY
C7E0:1C B1 INY
C7E1:1C B1 INY
C7E2:1C B1 INY
C7E3:1C B1 INY
C7E4:1C B1 INY
C7E5:1C B1 INY
C7E6:1C B1 INY
C7E7:1C B1 INY
C7E8:1C B1 INY
C7E9:1C B1 INY
C7EA:1C B1 INY
C7EB:1C B1 INY
C7EC:1C B1 INY
C7ED:1C B1 INY
C7EE:1C B1 INY
C7EF:1C B1 INY
C7F0:1C B1 INY
C7F1:1C B1 INY
C7F2:1C B1 INY
C7F3:1C B1 INY
C7F4:1C B1 INY
C7F5:1C B1 INY
C7F6:1C B1 INY
C7F7:1C B1 INY
C7F8:1C B1 INY
C7F9:1C B1 INY
C7FA:1C B1 INY
C7FB:1C B1 INY
C7FC:1C B1 INY
C7FD:1C B1 INY
C7FE:1C B1 INY
C7FF:1C B1 INY

```



```
C76A:  C76A 392 CALLSETUP EQU *
C76B:  C76B 393 CODECHMS
C76C:  C76C 394 LDA #PCOUNTS4.B
      :Setup some stuff for Prot Conv
      :Save cmd code for Prot Conv
      :# of parameters for cell
C76E:  C76E 396 CALLSETUP2 EQU *
C76F:  C76F 397 LDA #ARMNUM
C770:  C770 398 STA #PTALKUNIT
C771:  C771 399 STA NUMUNIT
C772:  C772 400 RTS
      :Alternate entry point
      :Move unit number to buff
      :Put in buffer
      :Back to caller
C776:  C776 402 APTALKCHMS EQU *
C777:  C777 403 DM ARINIT
C778:  C778 404 DM ARREADST
C779:  C779 405 DM ARWRITE
C780:  C780 406 DM ARSTATUS
C781:  C781 407 DM ARREADPROT
      :Appletalk init call
      :Appletalk readrest call
      :Appletalk write call
      :Appletalk status call
      :Appletalk readprot call
      :F82F INIT
C788:  C788 409 ROMSTUFFFILL EQU $C788-*
C789:  C789 410 DS ROMSTUFFFILL,$FF ;Fill character
C78B:  C78B 34 ELSE
C78C:  C78C 35 LST
C78D:  C78D 36 SKP 2
      :List by symbol end address
```

```
C9 ADDR# 00P
C742 APTALK2
C743 APTALK2
C5DB ARAPPLETALK2
C68A ARINIT1
C68B ARINIT2
C68C ARINIT3
C68D ARINIT4
C68E ARINIT5
C68F ARINIT6
C690 ARINIT7
C691 ARINIT8
C692 ARINIT9
C693 ARINIT10
C694 ARINIT11
C695 ARINIT12
C696 ARINIT13
C697 ARINIT14
C698 ARINIT15
C699 ARINIT16
C69A ARINIT17
C69B ARINIT18
C69C ARINIT19
C69D ARINIT20
C69E ARINIT21
C69F ARINIT22
C700 ARINIT23
C701 ARINIT24
C702 ARINIT25
C703 ARINIT26
C704 ARINIT27
C705 ARINIT28
C706 ARINIT29
C707 ARINIT30
C708 ARINIT31
C709 ARINIT32
C70A ARINIT33
C70B ARINIT34
C70C ARINIT35
C70D ARINIT36
C70E ARINIT37
C70F ARINIT38
C710 ARINIT39
C711 ARINIT40
C712 ARINIT41
C713 ARINIT42
C714 ARINIT43
C715 ARINIT44
C716 ARINIT45
C717 ARINIT46
C718 ARINIT47
C719 ARINIT48
C71A ARINIT49
C71B ARINIT50
C71C ARINIT51
C71D ARINIT52
C71E ARINIT53
C71F ARINIT54
C720 ARINIT55
C721 ARINIT56
C722 ARINIT57
C723 ARINIT58
C724 ARINIT59
C725 ARINIT60
C726 ARINIT61
C727 ARINIT62
C728 ARINIT63
C729 ARINIT64
C72A ARINIT65
C72B ARINIT66
C72C ARINIT67
C72D ARINIT68
C72E ARINIT69
C72F ARINIT70
C730 ARINIT71
C731 ARINIT72
C732 ARINIT73
C733 ARINIT74
C734 ARINIT75
C735 ARINIT76
C736 ARINIT77
C737 ARINIT78
C738 ARINIT79
C739 ARINIT80
C73A ARINIT81
C73B ARINIT82
C73C ARINIT83
C73D ARINIT84
C73E ARINIT85
C73F ARINIT86
C740 ARINIT87
C741 ARINIT88
C742 ARINIT89
C743 ARINIT90
C744 ARINIT91
C745 ARINIT92
C746 ARINIT93
C747 ARINIT94
C748 ARINIT95
C749 ARINIT96
C74A ARINIT97
C74B ARINIT98
C74C ARINIT99
C74D ARINIT100
C74E ARINIT101
C74F ARINIT102
C750 ARINIT103
C751 ARINIT104
C752 ARINIT105
C753 ARINIT106
C754 ARINIT107
C755 ARINIT108
C756 ARINIT109
C757 ARINIT110
C758 ARINIT111
C759 ARINIT112
C75A ARINIT113
C75B ARINIT114
C75C ARINIT115
C75D ARINIT116
C75E ARINIT117
C75F ARINIT118
C760 ARINIT119
C761 ARINIT120
C762 ARINIT121
C763 ARINIT122
C764 ARINIT123
C765 ARINIT124
C766 ARINIT125
C767 ARINIT126
C768 ARINIT127
C769 ARINIT128
C76A ARINIT129
C76B ARINIT130
C76C ARINIT131
C76D ARINIT132
C76E ARINIT133
C76F ARINIT134
C770 ARINIT135
C771 ARINIT136
C772 ARINIT137
C773 ARINIT138
C774 ARINIT139
C775 ARINIT140
C776 ARINIT141
C777 ARINIT142
C778 ARINIT143
C779 ARINIT144
C77A ARINIT145
C77B ARINIT146
C77C ARINIT147
C77D ARINIT148
C77E ARINIT149
C77F ARINIT150
C780 ARINIT151
C781 ARINIT152
C782 ARINIT153
C783 ARINIT154
C784 ARINIT155
C785 ARINIT156
C786 ARINIT157
C787 ARINIT158
C788 ARINIT159
C789 ARINIT160
C78A ARINIT161
C78B ARINIT162
C78C ARINIT163
C78D ARINIT164
C78E ARINIT165
C78F ARINIT166
C790 ARINIT167
C791 ARINIT168
C792 ARINIT169
C793 ARINIT170
C794 ARINIT171
C795 ARINIT172
C796 ARINIT173
C797 ARINIT174
C798 ARINIT175
C799 ARINIT176
C79A ARINIT177
C79B ARINIT178
C79C ARINIT179
C79D ARINIT180
C79E ARINIT181
C79F ARINIT182
C800 ARINIT183
C801 ARINIT184
C802 ARINIT185
C803 ARINIT186
C804 ARINIT187
C805 ARINIT188
C806 ARINIT189
C807 ARINIT190
C808 ARINIT191
C809 ARINIT192
C80A ARINIT193
C80B ARINIT194
C80C ARINIT195
C80D ARINIT196
C80E ARINIT197
C80F ARINIT198
C810 ARINIT199
C811 ARINIT200
C812 ARINIT201
C813 ARINIT202
C814 ARINIT203
C815 ARINIT204
C816 ARINIT205
C817 ARINIT206
C818 ARINIT207
C819 ARINIT208
C81A ARINIT209
C81B ARINIT210
C81C ARINIT211
C81D ARINIT212
C81E ARINIT213
C81F ARINIT214
C820 ARINIT215
C821 ARINIT216
C822 ARINIT217
C823 ARINIT218
C824 ARINIT219
C825 ARINIT220
C826 ARINIT221
C827 ARINIT222
C828 ARINIT223
C829 ARINIT224
C82A ARINIT225
C82B ARINIT226
C82C ARINIT227
C82D ARINIT228
C82E ARINIT229
C82F ARINIT230
C830 ARINIT231
C831 ARINIT232
C832 ARINIT233
C833 ARINIT234
C834 ARINIT235
C835 ARINIT236
C836 ARINIT237
C837 ARINIT238
C838 ARINIT239
C839 ARINIT240
C83A ARINIT241
C83B ARINIT242
C83C ARINIT243
C83D ARINIT244
C83E ARINIT245
C83F ARINIT246
C840 ARINIT247
C841 ARINIT248
C842 ARINIT249
C843 ARINIT250
C844 ARINIT251
C845 ARINIT252
C846 ARINIT253
C847 ARINIT254
C848 ARINIT255
C849 ARINIT256
C84A ARINIT257
C84B ARINIT258
C84C ARINIT259
C84D ARINIT260
C84E ARINIT261
C84F ARINIT262
C850 ARINIT263
C851 ARINIT264
C852 ARINIT265
C853 ARINIT266
C854 ARINIT267
C855 ARINIT268
C856 ARINIT269
C857 ARINIT270
C858 ARINIT271
C859 ARINIT272
C85A ARINIT273
C85B ARINIT274
C85C ARINIT275
C85D ARINIT276
C85E ARINIT277
C85F ARINIT278
C860 ARINIT279
C861 ARINIT280
C862 ARINIT281
C863 ARINIT282
C864 ARINIT283
C865 ARINIT284
C866 ARINIT285
C867 ARINIT286
C868 ARINIT287
C869 ARINIT288
C86A ARINIT289
C86B ARINIT290
C86C ARINIT291
C86D ARINIT292
C86E ARINIT293
C86F ARINIT294
C870 ARINIT295
C871 ARINIT296
C872 ARINIT297
C873 ARINIT298
C874 ARINIT299
C875 ARINIT300
C876 ARINIT301
C877 ARINIT302
C878 ARINIT303
C879 ARINIT304
C87A ARINIT305
C87B ARINIT306
C87C ARINIT307
C87D ARINIT308
C87E ARINIT309
C87F ARINIT310
C880 ARINIT311
C881 ARINIT312
C882 ARINIT313
C883 ARINIT314
C884 ARINIT315
C885 ARINIT316
C886 ARINIT317
C887 ARINIT318
C888 ARINIT319
C889 ARINIT320
C88A ARINIT321
C88B ARINIT322
C88C ARINIT323
C88D ARINIT324
C88E ARINIT325
C88F ARINIT326
C890 ARINIT327
C891 ARINIT328
C892 ARINIT329
C893 ARINIT330
C894 ARINIT331
C895 ARINIT332
C896 ARINIT333
C897 ARINIT334
C898 ARINIT335
C899 ARINIT336
C89A ARINIT337
C89B ARINIT338
C89C ARINIT339
C89D ARINIT340
C89E ARINIT341
C89F ARINIT342
C8A0 ARINIT343
C8A1 ARINIT344
C8A2 ARINIT345
C8A3 ARINIT346
C8A4 ARINIT347
C8A5 ARINIT348
C8A6 ARINIT349
C8A7 ARINIT350
C8A8 ARINIT351
C8A9 ARINIT352
C8AA ARINIT353
C8AB ARINIT354
C8AC ARINIT355
C8AD ARINIT356
C8AE ARINIT357
C8AF ARINIT358
C8B0 ARINIT359
C8B1 ARINIT360
C8B2 ARINIT361
C8B3 ARINIT362
C8B4 ARINIT363
C8B5 ARINIT364
C8B6 ARINIT365
C8B7 ARINIT366
C8B8 ARINIT367
C8B9 ARINIT368
C8BA ARINIT369
C8BB ARINIT370
C8BC ARINIT371
C8BD ARINIT372
C8BE ARINIT373
C8BF ARINIT374
C8C0 ARINIT375
C8C1 ARINIT376
C8C2 ARINIT377
C8C3 ARINIT378
C8C4 ARINIT379
C8C5 ARINIT380
C8C6 ARINIT381
C8C7 ARINIT382
C8C8 ARINIT383
C8C9 ARINIT384
C8CA ARINIT385
C8CB ARINIT386
C8CC ARINIT387
C8CD ARINIT388
C8CE ARINIT389
C8CF ARINIT390
C8D0 ARINIT391
C8D1 ARINIT392
C8D2 ARINIT393
C8D3 ARINIT394
C8D4 ARINIT395
C8D5 ARINIT396
C8D6 ARINIT397
C8D7 ARINIT398
C8D8 ARINIT399
C8D9 ARINIT400
C8DA ARINIT401
C8DB ARINIT402
C8DC ARINIT403
C8DD ARINIT404
C8DE ARINIT405
C8DF ARINIT406
C8E0 ARINIT407
C8E1 ARINIT408
C8E2 ARINIT409
C8E3 ARINIT410
C8E4 ARINIT411
C8E5 ARINIT412
C8E6 ARINIT413
C8E7 ARINIT414
C8E8 ARINIT415
C8E9 ARINIT416
C8EA ARINIT417
C8EB ARINIT418
C8EC ARINIT419
C8ED ARINIT420
C8EE ARINIT421
C8EF ARINIT422
C8F0 ARINIT423
C8F1 ARINIT424
C8F2 ARINIT425
C8F3 ARINIT426
C8F4 ARINIT427
C8F5 ARINIT428
C8F6 ARINIT429
C8F7 ARINIT430
C8F8 ARINIT431
C8F9 ARINIT432
C8FA ARINIT433
C8FB ARINIT434
C8FC ARINIT435
C8FD ARINIT436
C8FE ARINIT437
C8FF ARINIT438
C900 ARINIT439
C901 ARINIT440
C902 ARINIT441
C903 ARINIT442
C904 ARINIT443
C905 ARINIT444
C906 ARINIT445
C907 ARINIT446
C908 ARINIT447
C909 ARINIT448
C90A ARINIT449
C90B ARINIT450
C90C ARINIT451
C90D ARINIT452
C90E ARINIT453
C90F ARINIT454
C910 ARINIT455
C911 ARINIT456
C912 ARINIT457
C913 ARINIT458
C914 ARINIT459
C915 ARINIT460
C916 ARINIT461
C917 ARINIT462
C918 ARINIT463
C919 ARINIT464
C91A ARINIT465
C91B ARINIT466
C91C ARINIT467
C91D ARINIT468
C91E ARINIT469
C91F ARINIT470
C920 ARINIT471
C921 ARINIT472
C922 ARINIT473
C923 ARINIT474
C924 ARINIT475
C925 ARINIT476
C926 ARINIT477
C927 ARINIT478
C928 ARINIT479
C929 ARINIT480
C92A ARINIT481
C92B ARINIT482
C92C ARINIT483
C92D ARINIT484
C92E ARINIT485
C92F ARINIT486
C930 ARINIT487
C931 ARINIT488
C932 ARINIT489
C933 ARINIT490
C934 ARINIT491
C935 ARINIT492
C936 ARINIT493
C937 ARINIT494
C938 ARINIT495
C939 ARINIT496
C93A ARINIT497
C93B ARINIT498
C93C ARINIT499
C93D ARINIT500
C93E ARINIT501
C93F ARINIT502
C940 ARINIT503
C941 ARINIT504
C942 ARINIT505
C943 ARINIT506
C944 ARINIT507
C945 ARINIT508
C946 ARINIT509
C947 ARINIT510
C948 ARINIT511
C949 ARINIT512
C94A ARINIT513
C94B ARINIT514
C94C ARINIT515
C94D ARINIT516
C94E ARINIT517
C94F ARINIT518
C950 ARINIT519
C951 ARINIT520
C952 ARINIT521
C953 ARINIT522
C954 ARINIT523
C955 ARINIT524
C956 ARINIT525
C957 ARINIT526
C958 ARINIT527
C959 ARINIT528
C95A ARINIT529
C95B ARINIT530
C95C ARINIT531
C95D ARINIT532
C95E ARINIT533
C95F ARINIT534
C960 ARINIT535
C961 ARINIT536
C962 ARINIT537
C963 ARINIT538
C964 ARINIT539
C965 ARINIT540
C966 ARINIT541
C967 ARINIT542
C968 ARINIT543
C969 ARINIT544
C96A ARINIT545
C96B ARINIT546
C96C ARINIT547
C96D ARINIT548
C96E ARINIT549
C96F ARINIT550
C970 ARINIT551
C971 ARINIT552
C972 ARINIT553
C973 ARINIT554
C974 ARINIT555
C975 ARINIT556
C976 ARINIT557
C977 ARINIT558
C978 ARINIT559
C979 ARINIT560
C97A ARINIT561
C97B ARINIT562
C97C ARINIT563
C97D ARINIT564
C97E ARINIT565
C97F ARINIT566
C980 ARINIT567
C981 ARINIT568
C982 ARINIT569
C983 ARINIT570
C984 ARINIT571
C985 ARINIT572
C986 ARINIT573
C987 ARINIT574
C988 ARINIT575
C989 ARINIT576
C98A ARINIT577
C98B ARINIT578
C98C ARINIT579
C98D ARINIT580
C98E ARINIT581
C98F ARINIT582
C990 ARINIT583
C991 ARINIT584
C992 ARINIT585
C993 ARINIT586
C994 ARINIT587
C995 ARINIT588
C996 ARINIT589
C997 ARINIT590
C998 ARINIT591
C999 ARINIT592
C99A ARINIT593
C99B ARINIT594
C99C ARINIT595
C99D ARINIT596
C99E ARINIT597
C99F ARINIT598
C9A0 ARINIT599
C9A1 ARINIT600
C9A2 ARINIT601
C9A3 ARINIT602
C9A4 ARINIT603
C9A5 ARINIT604
C9A6 ARINIT605
C9A7 ARINIT606
C9A8 ARINIT607
C9A9 ARINIT608
C9AA ARINIT609
C9AB ARINIT610
C9AC ARINIT611
C9AD ARINIT612
C9AE ARINIT613
C9AF ARINIT614
C9B0 ARINIT615
C9B1 ARINIT616
C9B2 ARINIT617
C9B3 ARINIT618
C9B4 ARINIT619
C9B5 ARINIT620
C9B6 ARINIT621
C9B7 ARINIT622
C9B8 ARINIT623
C9B9 ARINIT624
C9BA ARINIT625
C9BB ARINIT626
C9BC ARINIT627
C9BD ARINIT628
C9BE ARINIT629
C9BF ARINIT630
C9C0 ARINIT631
C9C1 ARINIT632
C9C2 ARINIT633
C9C3 ARINIT634
C9C4 ARINIT635
C9C5 ARINIT636
C9C6 ARINIT637
C9C7 ARINIT638
C9C8 ARINIT639
C9C9 ARINIT640
C9CA ARINIT641
C9CB ARINIT642
C9CC ARINIT643
C9CD ARINIT644
C9CE ARINIT645
C9CF ARINIT646
C9D0 ARINIT647
C9D1 ARINIT648
C9D2 ARINIT649
C9D3 ARINIT650
C9D4 ARINIT651
C9D5 ARINIT652
C9D6 ARINIT653
C9D7 ARINIT654
C9D8 ARINIT655
C9D9 ARINIT656
C9DA ARINIT657
C9DB ARINIT658
C9DC ARINIT659
C9DD ARINIT660
C9DE ARINIT661
C9DF ARINIT662
C9E0 ARINIT663
C9E1 ARINIT664
C9E2 ARINIT665
C9E3 ARINIT666
C9E4 ARINIT667
C9E5 ARINIT668
C9E6 ARINIT669
C9E7 ARINIT670
C9E8 ARINIT671
C9E9 ARINIT672
C9EA ARINIT673
C9EB ARINIT674
C9EC ARINIT675
C9ED ARINIT676
C9EE ARINIT677
C9EF ARINIT678
C9F0 ARINIT679
C9F1 ARINIT680
C9F2 ARINIT681
C9F3 ARINIT682
C9F4 ARINIT683
C9F5 ARINIT684
C9F6 ARINIT685
C9F7 ARINIT686
C9F8 ARINIT687
C9F9 ARINIT688
C9FA ARINIT689
C9FB ARINIT690
C9FC ARINIT691
C9FD ARINIT692
C9FE ARINIT693
C9FF ARINIT694
C900 ARINIT695
C901 ARINIT696
C902 ARINIT697
C903 ARINIT698
C904 ARINIT699
C905 ARINIT700
C906 ARINIT701
C907 ARINIT702
C908 ARINIT703
C909 ARINIT704
C90A ARINIT705
C90B ARINIT706
C90C ARINIT707
C90D ARINIT708
C90E ARINIT709
C90F ARINIT710
C910 ARINIT711
C911 ARINIT712
C912 ARINIT713
C913 ARINIT714
C914 ARINIT715
C915 ARINIT716
C916 ARINIT717
C917 ARINIT718
C918 ARINIT719
C919 ARINIT720
C91A ARINIT721
C91B ARINIT722
C91C ARINIT723
C91D ARINIT724
C91E ARINIT725
C91F ARINIT726
C920 ARINIT727
C921 ARINIT728
C922 ARINIT729
C923 ARINIT730
C924 ARINIT731
C925 ARINIT732
C926 ARINIT733
C927 ARINIT734
C928 ARINIT735
C929 ARINIT736
C92A ARINIT737
C92B ARINIT738
C92C ARINIT739
C92D ARINIT740
C92E ARINIT741
C92F ARINIT742
C930 ARINIT743
C931 ARINIT744
C932 ARINIT745
C933 ARINIT746
C934 ARINIT747
C935 ARINIT748
C936 ARINIT749
C937 ARINIT750
C938 ARINIT751
C939 ARINIT752
C93A ARINIT753
C93B ARINIT754
C93C ARINIT755
C93D ARINIT756
C93E ARINIT757
C93F ARINIT758
C940 ARINIT759
C941 ARINIT760
C942 ARINIT761
C943 ARINIT762
C944 ARINIT763
C945 ARINIT764
C946 ARINIT765
C947 ARINIT766
C948 ARINIT767
C949 ARINIT768
C94A ARINIT769
C94B ARINIT770
C94C ARINIT771
C94D ARINIT772
C94E ARINIT773
C94F ARINIT774
C950 ARINIT775
C951 ARINIT776
C952 ARINIT777
C953 ARINIT778
C954 ARINIT779
C955 ARINIT780
C956 ARINIT781
C957 ARINIT782
C958 ARINIT783
C959 ARINIT784
C95A ARINIT785
C95B ARINIT786
C95C ARINIT787
C95D ARINIT788
C95E ARINIT789
C95F ARINIT790
C960 ARINIT791
C961 ARINIT792
C962 ARINIT793
C963 ARINIT794
C964 ARINIT795
C965 ARINIT796
C966 ARINIT797
C967 ARINIT798
C968 ARINIT799
C969 ARINIT800
C96A ARINIT801
C96B ARINIT802
C96C ARINIT803
C96D ARINIT804
C96E ARINIT805
C96F ARINIT806
C970 ARINIT807
C971 ARINIT808
C972 ARINIT809
C973 ARINIT810
C974 ARINIT811
C975 ARINIT812
C976 ARINIT813
C977 ARINIT814
C978 ARINIT815
C979 ARINIT816
C97A ARINIT817
C97B ARINIT818
C97C ARINIT819
C97D ARINIT820
C97E ARINIT821
C97F ARINIT822
C980 ARINIT823
C981 ARINIT824
C982 ARINIT825
C983 ARINIT826
C984 ARINIT827
C985 ARINIT828
C986 ARINIT829
C987 ARINIT830
C988 ARINIT831
C989 ARINIT832
C98A ARINIT833
C98B ARINIT834
C98C ARINIT835
C98D ARINIT836
C98E ARINIT837
C98F ARINIT838
C990 ARINIT839
C991 ARINIT840
C992 ARINIT841
C993 ARINIT842
C994 ARINIT843
C995 ARINIT844
C996 ARINIT845
C997 ARINIT846
C998 ARINIT847
C999 ARINIT848
C99A ARINIT849
C99B ARINIT850
C99C ARINIT851
C99D ARINIT852
C99E ARINIT853
C99F ARINIT854
C9A0 ARINIT855
C9A1 ARINIT856
C9A2 ARINIT857
C9A3 ARINIT858
C9A4 ARINIT859
C9A5 ARINIT860
C9A6 ARINIT861
C9A7 ARINIT862
C9A8 ARINIT863
C9A9 ARINIT864
C9AA ARINIT865
C9AB ARINIT866
C9AC ARINIT867
C9AD ARINIT868
C9AE ARINIT869
C9AF ARINIT870
C9B0 ARINIT871
C9B1 ARINIT872
C9B2 ARINIT873
C9B3 ARINIT874
C9B4 ARINIT875
C9B5 ARINIT876
C9B6 ARINIT877
C9B7 ARINIT878
C9B8 ARINIT879
C9B9 ARINIT880
C9BA ARINIT881
C9BB ARINIT882
C9BC ARINIT883
C9BD ARINIT884
C9BE ARINIT885
C9BF ARINIT886
C9C0 ARINIT887
C9C1 ARINIT888
C9C2 ARINIT889
C9C3 ARINIT890
C9C4 ARINIT891
C9C5 ARINIT892
C9C6 ARINIT893
C9C7 ARINIT894
C9C8 ARINIT895
C9C9 ARINIT896
C9CA ARINIT897
C9CB ARINIT898
C9CC ARINIT899
C9CD ARINIT900
C9CE ARINIT901
C9CF ARINIT902
C9D0 ARINIT903
C9D1 ARINIT904
C9D2 ARINIT905
C9D3 ARINIT906
C9D4 ARINIT907
C9D5 ARINIT908
C9D6 ARINIT909
C9D7 ARINIT910
C9D8 ARINIT911
C9D9 ARINIT912
C9DA ARINIT913
C9DB ARINIT914
C9DC ARINIT915
C9DD ARINIT916
C9DE ARINIT917
C9DF ARINIT918
C9E0 ARINIT919
C9E1 ARINIT920
C9E2 ARINIT921
C9E3 ARINIT922
C9E4 ARINIT923
C9E5 ARINIT924
C9E6 ARINIT925
C9E7 ARINIT926
C9E8 ARINIT927
C9E9 ARINIT928
C9EA ARINIT929
C9EB ARINIT930
C9EC ARINIT931
C9ED ARINIT932
C9EE ARINIT933
C9EF ARINIT934
C9F0 ARINIT935
C9F1 ARINIT936
C9F2 ARINIT937
C9F3 ARINIT938
C9F4 ARINIT939
C9F5 ARINIT940
C9F6 ARINIT941
C9F7 ARINIT942
C9F8 ARINIT943
C9F9 ARINIT944
C9FA ARINIT945
C9FB ARINIT946
C9FC ARINIT947
C9FD ARINIT948
C9FE ARINIT949
C9FF ARINIT950
C900 ARINIT951
C901 ARINIT952
C902 ARINIT953
C903 ARINIT954
C904 ARINIT955
C905 ARINIT956
C906 ARINIT957
C907 ARINIT958
C908 ARINIT959
C909 ARINIT960
C90A ARINIT961
C90B ARINIT962
C90C ARINIT963
C90D ARINIT964
C90E ARINIT965
C90F ARINIT966
C910 ARINIT967
C911 ARINIT968
C912 ARINIT969
C913 ARINIT970
C914 ARINIT971
C915 ARINIT972
C916 ARINIT973
C917 ARINIT974
C918 ARINIT975
C919 ARINIT976
C91A ARINIT977
C91B ARINIT978
C91C ARINIT979
C91D ARINIT980
C91E ARINIT981
C91F ARINIT982
C920 ARINIT983
C921 ARINIT984
C922 ARINIT985
C923 ARINIT986
C924 ARINIT987
C925 ARINIT988
C926 ARINIT989
C927 ARINIT990
C928 ARINIT991
C929 ARINIT992
C92A ARINIT993
C92B ARINIT994
C92C ARINIT995
C92D ARINIT996
C92E ARINIT997
C92F ARINIT998
C930 ARINIT999
C931 ARINIT1000
C932 ARINIT1001
C933 ARINIT1002
C934 ARINIT1003
C935 ARINIT1004
C936 ARINIT1005
C937 ARINIT1006
C938 ARINIT1007
C939 ARINIT1008
C93A ARINIT1009
C93B ARINIT1010
C93C ARINIT1011
C93D ARINIT1012
C93E ARINIT1013
C93F ARINIT1014
C940 ARINIT1015
C941 ARINIT1016
C942 ARINIT1017
C943 ARINIT1018
C944 ARINIT1019
C945 ARINIT1020
C946 ARINIT1021
C947 ARINIT1022
C948 ARINIT1023
C949 ARINIT1024
C94A ARINIT1025
C94B ARINIT1026
C94C ARINIT1027
C94D ARINIT1028
C94E ARINIT1029
C94F ARINIT1030
C950 ARINIT1031
C951 ARINIT1032
C952 ARINIT1033
C953 ARINIT1034
C954 ARINIT1035
C955 ARINIT1036
C956 ARINIT1037
C957 ARINIT1038
C958 ARINIT1039
C959 ARINIT1040
C95A ARINIT1041
C95B ARINIT1042
C95C ARINIT1043
C95D ARINIT1044
C95E ARINIT1045
C95F ARINIT1046
C960 ARINIT1047
C961 ARINIT1048
C962 ARINIT1049
C963 ARINIT1050
C964 ARINIT1051
C965 ARINIT1052
C966 ARINIT1053
C967 ARINIT1054
C968 ARINIT1055
C969 ARINIT1056
C96A ARINIT1057
C96B ARINIT1058
C96C ARINIT1059
C96D ARINIT1060
C96E ARINIT1061
C96F ARINIT1062
C970 ARINIT1063
C971 ARINIT1064
C972 ARINIT1065
C973 ARINIT1066
C974 ARINIT1067
C975 ARINIT1068
C976 ARINIT1069
C977 ARINIT1070
C978 ARINIT1071
C979 ARINIT1072
C97A ARINIT1073
C97B ARINIT1074
C97C ARINIT1075
C97D ARINIT1076
C97E ARINIT1077
C97F ARINIT1078
C980 ARINIT1079
C981 ARINIT1080
C982 ARINIT1081
C983 ARINIT1082
C984 ARINIT1083
C985 ARINIT1084
C986 ARINIT1085
C987 ARINIT1086
C988 ARINIT1087
C989 ARINIT1088
C98A ARINIT1089
C98B ARINIT1090
C98C ARINIT1091
C98D ARINIT1092
C98E ARINIT1093
C98F ARINIT1094
C990 ARINIT1095
C991 ARINIT1096
C992 ARINIT1097
C993 ARINIT1098
C994 ARINIT1099
C995 ARINIT1100
C996 ARINIT1101
C997 ARINIT1102
C998 ARINIT1103
C999 ARINIT1104
C99A ARINIT1105
C99B ARINIT1106
C99C ARINIT1107
C99D ARINIT1108
C99E ARINIT1109
C99F ARINIT1110
C9A0 ARINIT1111
C9A1 ARINIT1112
C9A2 ARINIT1113
C9A3 ARINIT1114
C9A4 ARINIT1115
C9A5 ARINIT1116
C9A6 ARINIT1117
C9A7 ARINIT1118
C9A8 ARINIT1119
C9A9 ARINIT1120
C9AA ARINIT1121
C9AB ARINIT1122
C9AC ARINIT1123
C9AD ARINIT1124
C9AE ARINIT1125
C9AF ARINIT1126
C9B0 ARINIT1127
C9B1 ARINIT1128
C9B2 ARINIT1129
C9B3 ARINIT1130
C9B4 ARINIT1131
C9B5 ARINIT1132
C9B6 ARINIT1133
C9B7 ARINIT1134
C9B8 ARINIT1135
C9B9 ARINIT1136
C9BA ARINIT1137
C9BB ARINIT1138
C9BC ARINIT1139
C9BD ARINIT1140
C9BE ARINIT1141
C9BF ARINIT1142
C9C0 ARINIT1143
C9C1 ARINIT1144
C9C2 ARINIT1145
C9C3 ARINIT1146
C9C4 ARINIT1147
C9C5 ARINIT1148
C9C6 ARINIT1149
C9C7 ARINIT1150
C9C8 ARINIT1151
C9C9 ARINIT1152
C9CA ARINIT1153
C9CB ARINIT1154
C9CC ARINIT1155
C9CD ARINIT1156
C9CE ARINIT1157
C9CF ARINIT1158
C9D0 ARINIT1159
C9D1 ARINIT1160
C9D2 ARINIT1161
C9D3 ARINIT1162
```

Glossary

65C02: The microprocessor used in the Apple IIc computer.

ACIA: Asynchronous communications interface adapter. A single chip that converts data from parallel to serial form, and vice versa, and handles serial transmission and reception and RS-232-C signals, under the control of its internal registers set and changed by firmware or software.

accumulator: The register in the 6502 and 65C02 microprocessors where most computations are performed.

acronym: A word formed from the initial letters of a name or phrase, such as ROM, from read-only memory.

ADC: See **analog-to-digital converter**.

address: A number used to identify something, such as a location in the computer's memory.

analog: Represented in terms of a physical quantity that can vary smoothly and continuously over a range of values. For example, a conventional 12-hour clock face is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

analog-to-digital converter: A device that converts quantities from analog to digital form. For example, the Apple IIc's hand controller converts the position of the controller dial (an analog quantity) into a discrete number (a digital quantity) that changes in steps even when the dial is turned smoothly.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR**, **exclusive OR**, **NOT**.

Applesoft: An extended version of the BASIC programming language used with the Apple IIc computer. The firmware for interpreting and executing programs in Applesoft is included in the Apple IIc ROM.

ASCII: American Standard Code for Information Interchange; a code in which the numbers from 0 to 127 stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

asserted: Made true (positive in positive-true logic; negative in negative-true logic).

asynchronous: Having a variable time interval between characters.

back panel: The rear face of the Apple IIc computer, which includes the power switch, the power connector, and connectors for two serial devices, a video display device, an external disk drive, and a mouse or hand control.

bandwidth: A measure of the range of frequencies a device can handle. In the case of a video monitor, greater bandwidth enables it to display more information; to display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

base address: In indexed addressing, the fixed component of an address.

baud: A unit of signaling speed equal to the number of discrete conditions or signal events per second. Often equated (though not precisely) with bits per second.

binary: The representation of numbers in terms of powers of 2, using the two digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen.

bit: A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

board: See **printed-circuit board**.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to *pull itself in by its own bootstraps*.

bootstrap: See **boot**.

BREAK: A SPACE (0) signal sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a timesharing service.

BRK: A 65C02 instruction that causes the microprocessor to halt.

buffer: An area of the computer's memory used as a holding area where information can be stored by one program or device and then read out by another at a different speed.

bus: A group of wires that transmit related information from one part of a computer system to another. In the Apple IIc, the address bus has 16 wires, and the data bus has 8.

byte: A unit of information consisting of a fixed number of bits; on the Apple IIc, one byte consists of eight bits and can represent any value between 0 and 255.

carriage return: An ASCII character (decimal 13; Appendix H) that ordinarily causes a printer or display device to place the subsequent character on the left margin. On a manual typewriter, this movement is combined with line feed (the advancement of the paper to the next line). With computers, carriage return and line feed are separate, causing hair-raising problems for the user.

carrier: The background signal on a communication channel that is modified to *carry* the information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

carry flag: The C bit in the 65C02 processor status register, used to hold the carry bit in addition and subtraction.

cathode-ray tube: An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

central processing unit: See **processor**.

character: A letter, digit, punctuation mark, or other symbol used in printing, displaying or transferring information.

character code: A number used to represent a text character for processing by a computer system.

chip: The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated.

Clear To Send: An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

cold start: The process of starting up the Apple IIc when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, then loading and running a program. Compare **warm start**.

command: A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some action.

command character: An ASCII character, usually CONTROL-A or CONTROL-I, that causes the serial port firmware to interpret subsequent characters as a command.

command register: An ACIA location (at address \$C09A for port 1 and \$C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

communication mode: An operating state in which serial port 2 (or 1, if so set) is prepared to exchange data and signals with a DCE (such as a modem).

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare **interpreter**.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it.

computer: An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

computer system: A computer and its associated hardware, firmware, and software.

connector: A physical device such as a plug, socket, or jack, used to connect two devices to one another.

control character: A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between \$00 and \$1F (or between \$80 and \$9F if the high-order bit is set)). You can generate them at the Apple IIc keyboard by holding down the **CONTROL** key while typing one of the letter keys or [\] or _.

control register: An ACIA location (at address \$C09B for port 1, or \$C0AB for port 2) that stores data format and baud rate selections.

CPU: Central processing unit; see **processor**.

CRT: See **cathode-ray tube**.

CTS: See **Clear To Send**.

cursor: A symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter**.

data: Information; especially information used or operated on by a program.

data bit: One of five to eight bits representing a character.

Data Carrier Detect: An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established.

Data Communication Equipment: As defined by the RS-232-C standard, any device that transmits or receives information. Usually this is a modem. However, when a Modem Eliminator is used, the Apple IIc itself looks like a DCE to the other device, and the other device looks like a DCE to the Apple IIc.

data format: The form in which data are stored, manipulated, or transferred. Serial data transmitted and received by port 1 or 2 have a data format of: one start bit, five to eight data bits, an optional parity bit, and one, one and a half, or two stop bits.

Data Set Ready: An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

Data Terminal Equipment: As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as a terminus of a communication connection.

Data Terminal Ready: An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

DCD: See **Data Carrier Detect**.

DCE: See **Data Communication Equipment**.

debug: To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems.

decimal: The common form of number representation in everyday usage, in which numbers are expressed in terms of powers of 10, using the ten digits 0 to 9.

default: A value, action, or setting that is assumed or set in the absence of explicit instructions otherwise.

demodulate: To recover the information being transmitted by a modulated signal; for example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into sound emitted by a speaker.

device: (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

digit: (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

digital: Represented in a discrete (noncontinuous) form, such as numerical digits. For example, contemporary digital clocks display the time in numerical form (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog**.

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package**.

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. The opposite of an assembler.

disk: An information storage medium consisting of a flat, circular magnetic surface on which information can be recorded in the form of small magnetized spots, similarly to the way sounds are recorded on tape.

disk drive: A device that writes and reads information on the surface of a magnetic disk.

diskette: A term sometimes used for the small (5¼-inch) flexible disks used with the Apple Disk II drive.

Disk IIc drive: A model of disk drive made and sold by Apple Computer for use with the Apple IIc computer; uses 5¼-inch flexible (floppy) disks.

Disk Operating System: An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

display: (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

display device: A device that exhibits information visually, such as a television receiver or video monitor.

display screen: The glass or plastic panel on the front of a display device, on which images are displayed.

DOS: See **Disk Operating System**.

DSR: See **Data Set Ready**.

DTE: See **Data Terminal Equipment**.

DTR: See **Data Terminal Ready**.

dual in-line package: An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side; similar in appearance to an armored centipede.

echo: To send an input character to a video display, printer, or other output device.

edit: To change or modify; for example, to insert, remove, replace, or move text in a document.

editor: A program that enables the user to create and edit information of a particular form; for example, a *text editor* or a *graphics editor*.

effective address: In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

emulation mode: A manner of operating in which one computer or interface imitates another.

even parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits (among the data bits plus the parity bit) an even number.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.

escape mode: A state of the Apple IIc computer, entered by pressing the **[ESC]** key, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes, beginning with `[ESC]`, used for positioning the cursor and controlling the display of text on the screen.

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false; compare **OR**, **AND**, **NOT**.

execute: To perform or carry out a specified action or sequence of actions, such as those described by a program.

firmware: Software stored permanently in hardware: programs in read-only memory (ROM). Such programs (for example, the Applesoft interpreter and the Apple IIc Monitor program) are built into the computer at the factory; they can be executed at any time but cannot be modified or erased from main memory. Compare **hardware**, **software**.

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Compare **floating-point**.

flexible disk: A disk made of flexible plastic; often called a *floppy* disk. Compare **rigid disk**.

floating-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to *float* to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

form feed: An ASCII character (decimal 12; Appendix H) that causes a printer or other paper-handling device to advance to the top of the next page.

framing error: In serial data transfer, absence of the expected stop bit(s) at the end of a received character. The serial port 1 and 2 ACIAs record this error by setting bit 1 (FRM) of its status register to 1. The ACIA checks and records each framing error separately: if the next character is OK, the FRM bit is cleared.

full duplex: Capable of simultaneous two-way communication.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

half duplex: Capable of communication in one direction at a time.

hand controller: An optional peripheral device that can be connected to the Apple IIc's hand controller connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

hand controller connector: A 9-pin connector on the Apple IIc's back panel, used for connecting hand controllers to the computer.

hardware: Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software**, **firmware**.

hertz: The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicist Heinrich Hertz and abbreviated Hz. The Apple IIc's 65C02 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

high-resolution graphics: The display of graphics on the Apple IIc's display screen as a six-color array of points, 280 columns wide and 192 rows high.

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off; compare **setup time**.

Hz: See **hertz**.

IC: See **integrated circuit**.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

indexed addressing: A method of specifying memory addresses used in machine-language programming.

index register: A register in a computer processor that holds an index for use in indexed addressing. The Apple IIc's 65C02 microprocessor has two index registers, called the X register and the Y register.

input: (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number, with no fractional part; represented inside the computer in fixed-point form. Compare **real number**.

integrated circuit: An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see **chip**.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interpreter: A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare **compiler**.

interrupt: A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

inverse video: The display of text on the computer's display screen in the form of black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

I/O: Input/output; the transfer of information into and out of a computer. See **input, output**.

I/O device: Input/output device; a device that transfers information into or out of a computer. See **input, output, peripheral device**.

I/O link: A fixed location that contains the address of an input/output subroutine in the Apple IIc Monitor program.

K: Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIc computer, similar to a typewriter keyboard, for typing information to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as **CONTROL**-**C**) on the Apple IIc keyboard.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see **K**.

KSW: The symbolic name of the location in the Apple IIc's memory where the standard input link is stored; stands for *keyboard switch*. See **I/O link**.

language: See **programming language**.

language translator: A system program that reads a program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter, compiler, assembler**.

least significant bit: The right-hand bit of a binary number as written down; its positional value is 0 or 1.

line feed: An ASCII character (decimal 10; Appendix H) that ordinarily causes a printer or video display to advance to the next line.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

local: Nearby; capable of direct connection using wires only.

location: See **memory location**.

logical operator: An operator, such as **AND**, that combines logical values to produce a logical result.

low-level language: A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIc include 65C02 machine language and 65C02 assembly language.

low-order byte: The less significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

low-power Schottky: A type of TTL integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.

low-resolution graphics: The display of graphics on the Apple IIc's display screen as a 16-color array of blocks, 40 columns wide and 48 rows high.

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 65C02 microprocessor used in the Apple IIc) has its own form of machine language.

main memory: The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device can then check for errors by looking for this value on each character.

memory: A hardware component of a computer system that can store information for later retrieval; see **main memory**, **random-access memory**, **read-only memory**, **read-write memory**.

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple IIc, a memory location holds one byte, or eight bits, of information.

MHz: Megahertz; one million hertz. See **hertz**.

microcomputer: A computer, such as the Apple IIc, whose processor is a microprocessor.

microprocessor: A computer processor contained in a single integrated circuit, such as the 65C02 microprocessor used in the Apple IIc.

microsecond: One millionth of a second; abbreviated μ s.

millisecond: One thousandth of a second; abbreviated ms.

mode: A state of a computer or system that determines its behavior.

modem: Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line; a DCE that connects a DTE to communication lines.

modem eliminator: The physical crossing of wires that replaces a pair of modems for direct connection of two DTEs.

modulate: To modify or alter a signal so as to transmit information; for example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or AM) or the frequency (frequency modulation, or FM) of a carrier signal.

monitor: See **video monitor**.

Monitor program: A system program built into the Apple IIc in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

most significant bit: The leftmost bit of a binary number as written down. This bit represents 0 or 1 times 2 to the power one less than the total number of bits in the binary number. For example, in the binary number 10000, which contains five digits, the 1 represents 1 times 2 to the 4th power—or 16.

nanosecond: One billionth (in British usage, one thousand-millionth) of a second; abbreviated ns.

network: A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them.

nibble: A unit of information equal to half a byte, or four bits; can hold any value from 0 to 15. Sometimes spelled *nybble*.

NOT: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare **AND**, **OR**, **exclusive OR**.

NTSC: (1) National Television Standards Committee; the committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

object code: See **object program**.

object program: The translated form of a program produced by a language translator such as a compiler or assembler; also called object code. Compare **source program**.

odd parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1-bits an odd number.

opcode: See **operation code**.

operand: A value to which an operator is applied; the value on which an opcode operates.

operating system: A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

operation code: The part of a machine-language instruction that specifies the operation to be performed; often called **opcode**.

operator: A symbol or sequence of characters, such as + or *AND*, specifying an operation to be performed on one or more values (the operands) to produce a result.

OR: A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false; compare **exclusive OR**, **AND**, **NOT**.

output: Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

overflow: A condition that occurs when the Apple IIc processor does not retrieve a received character from the ACIA's receive data register before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

page: (1) A screenful of information on a video display, consisting on the Apple IIc of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes.

page zero: See **zero page**.

parallel interface: An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

parity: Maintenance of a sameness of level or count, usually the count of 1-bits in each character, for error checking.

parity error: Absence of the correct parity bit value in a received character. The serial port ACIAs record this error by setting bit 0 (PAR) of their status registers to 1.

PC board: See **printed-circuit board**.

phase: (1) A stage in a periodic process; a point in a cycle; for example, the 65C02 microprocessor uses a clock cycle consisting of two phases called $\phi 0$ and $\phi 1$. (2) The relationship between two periodic signals or processes; for example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

pointer: An item of information consisting of the memory address of some other item.

pop: To remove the top entry from a stack.

port: The point of connection, usually a physical connector, between a computer and a peripheral device, another computer, or a network.

power supply: The hardware component of a computer that draws electrical power from a power outlet and converts it to the forms needed by some other hardware component.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, from which all conducting material except the desired circuits is etched, and to which integrated circuits and other electronic components are connected.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

program: (1) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (2) To write a program.

programming language: A set of rules or conventions for writing programs.

prompt: To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character: A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character **J** is used by the Applesoft BASIC interpreter, **>** by Integer BASIC, and ***** by the System Monitor program.

prompt message: A message displayed on the screen to prompt the user for some action.

protocol: A predefined exchange of control signals between devices enabling them to prepare for and carry out coordinated data transfers.

push: To add an entry to the top of a stack.

radio-frequency modulator: A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

RAM: See **random-access memory**.

random-access memory: Memory in which the contents of individual locations can be referred to in an arbitrary or random order.

raster: The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive dots on the individual lines of the raster.

read: To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory: Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare **read-write memory, random-access memory, write-only memory**.

read-write memory: Memory whose contents can be both read and written; often misleadingly called *random-access memory*, or *RAM*. The information contained in read-write memory is erased when the computer's power is turned off, and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare **read-only memory, random-access memory, write-only memory**.

receive data register: A read-only register in each serial port ACIA (at location \$C098 for port 1 and \$C0A8 for port 2) that stores the most recent character successfully received.

register: A location in a computer processor where an item of information, such as a byte, is held and modified under program control. Registers in the 65C02 microprocessor include the accumulator (A), two index registers (X and Y), the stack pointer (S), the processor status register (P), and the program counter (PC). The PC register holds two bytes (16 bits); the other registers hold one byte (8 bits) each.

remote: Too distant for direct connection using wires or cables only.

Request To Send: An RS-232-C signal from a DTE to a DCE to prepare the DCE for data transmission.

return address: The point in a program to which control returns on completion of a subroutine.

RF modulator: See **radio-frequency modulator**.

RI: See **Ring Indicator**.

rigid disk: A disk made of a hard, nonflexible material. Compare **flexible disk**.

Ring Indicator: An optional RS-232-C signal from a DCE to a DTE that indicates the arrival of a call.

ROM: See **read-only memory**.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

RS-232-C: A standard created by the Electronic Industries Association (EIA) to allow devices of different manufacturers to exchange serial data—particularly via telephone lines.

RTS: See **Request To Send**.

run: (1) To execute a program.
(2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

screen: See **display screen**.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **viewport, window**.

serial interface: An interface in which information is transmitted sequentially, one bit at a time, over a single wire or channel. Compare **parallel interface**.

setup time: The amount of time a signal must be valid in advance of some event; compare **hold time**.

silicon: A nonmetallic, semiconducting chemical element from which integrated circuits are made.

soft switch: A means of changing some feature of the Apple IIc from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

software: Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware, firmware**.

source code: See **source program**.

source program: The original form of a program given to a language translator such as a compiler or assembler for conversion into another form; sometimes called *source code*. Compare **object program**.

space character: A text character whose printed representation is a blank space, typed from the keyboard by pressing the SPACE bar.

SPACE parity: A bit of value 0 appended to a binary number for transmission. The receiving device can look for this value on each character as a means of error checking.

stack: A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order.

start bit: A transition from a MARK signal to a SPACE signal for one bit-time, indicating that the next string of bits represents a character.

status register: A register in an ACIA (at location \$C099 for port 1 and \$C0A9 for port 2) that stores the state of two of the RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

stop bit: A MARK signal following a string of data bits (or their optional parity bit) to indicate the end of a character.

string: An item of information consisting of a sequence of text characters.

strobe: (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

television receiver: A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIc computer. Compare **video monitor**.

terminal: A device consisting of a typewriterlike keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as the Apple IIc typically have all or part of a terminal built into them.

terminal mode: An operating state of the Apple IIc communication port in which the firmware makes the computer act like a simple ASCII terminal.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple IIc's display screen. Compare **graphics**.

text window: An area on the Apple IIc's display screen within which text is displayed and scrolled.

transistor-to-transistor logic:

(1) A family of integrated circuits used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical 0s and 1s.

transmit data register: A write-only register in one of the serial port ACIAs (at location \$C098 for port 1 and \$C0A8 for port 2) that holds the current character to be transmitted.

troubleshoot: To locate and correct the cause of a problem or malfunction in a computer system. Typically used to refer to hardware-related problems; compare **debug**.

TTL: See **transistor-to-transistor logic**.

unary operator: An operator that applies to a single operand; for example, the minus sign (—) in a negative number such as —6 is a unary arithmetic operator.

user: The person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

vector: (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple IIc computer as a display device. Compare **television receiver**.

viewport: All or part of the display screen, used by an application program to display a portion of the information (such as a document, picture, or worksheet) that the program is working on. Compare **window**.

warm start: The process of restarting the Apple IIc after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory. Compare **cold start**.

window: The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen; compare **viewport**.

word: A group of bits of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

wraparound: The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

X register: One of the index registers in the 65C02 microprocessor.

Y register: One of the index registers in the 65C02 microprocessor.

zero page: The first page (256 bytes) of the Apple IIc's memory, also called page \$00. Since the high-order byte of any address in this page is 0, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.

Bibliography

Apple II Monitors Peeled. Cupertino, Calif.: Apple Computer, Inc., 1978.

Currently not updated for Apple IIe and IIfx, but a good introduction to Apple II series input/output procedures; also useful for historical background.

Apple IIe Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1982.

Addendum to the Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1984.

Applesoft BASIC Programmer's Reference Manual, Volumes 1 and 2.

For the Apple II, IIe, and IIfx. Cupertino, Calif.: Apple Computer, Inc., 1982.

The version that applies to both the Apple IIe and the Apple IIfx has Apple product number A2L0084 (Vol. 1) and A2L0085 (Vol. 2).

Applesoft Tutorial. Cupertino, Calif.: Apple Computer, Inc., 1982.

Leventhal, Lance. *6502 Assembly Language Programming*. Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

Synertek Hardware Manual. Santa Clara, Calif.: Synertek Incorporated, 1976.

Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500 series microcomputers.







Synertek Programming Manual. Santa Clara, Calif.: Synertek, Incorporated, 1976.

The only currently available manufacturer's programming manual for 6500 series microcomputers.

- Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte* Vol. 5, No. 11 (November 1980).
- . "A Simplified Theory of Video Graphics, Part II." *Byte* Vol. 5, No. 12 (December 1980).
- . "More Colors for Your Apple." *Byte* Vol. 4, No. 6 (June 1979).
- . "True Sixteen-Color Hi-Res." *Apple Orchard* Vol. 5, No. 1 (January 1984).
- Wozniak, Steve. "System Description: The Apple II." *Byte* Vol. 2, No. 5 (May 1977).
- . "SWEET16: The 6502 Dream Machine." *Byte* Vol. 2, No. 10 (October 1977).

Index

Cast of Characters

- * (asterisk) 55, 188
- \ (backslash) 55, 58
- > (greater-than sign) 55
- ? (question mark) 55
-] (right bracket) 55
-  4, 153
-  4, 58
-  4, 59
-  4
-  4, 50, 76
-  4, 50, 76

A

- A register 16
- accumulator 16
- ACIA
 - block diagram 259
 - buffering 257
 - command register 263
 - control register 261–262
 - interrupts 334
 - register locations for port 1 149
 - register locations for port 2 161
 - status register 264
 - transmit/receive register 265
- address space switches 21–22
- address transformation display 242
- addresses
 - auxiliary RAM 20
 - display page 94
 - hardware 21–22
 - I/O link 52–54
 - main RAM 20
 - memory 18
 - ROM 20

- addressing modes 24, 211
- AKD (any-key-down) 72, 227–228
- AltChar switch 95
- alternate character set 64, 65, 82
- AltZP switch 25, 26
- analog inputs 185, 186, 270
- any-key-down (AKD) 72, 227–228
- Apple II series
 - comparisons among 336–352
 - ID bytes 338
 - port I/O differences 66–69
- Applesoft BASIC 317
 - prompt character for 55
- Applesoft BASIC interpreter 13
 - addresses 234
 - and line length 56
 - firmware addresses 21
- arithmetic, hexadecimal 199
- arrow keys 4, 56
- ASCII character set
 - and MouseText 84
 - code conversions 381–384
 - code equivalents 370–371
 - codes 74–75
- ASCII input mode 193
- asterisk (*) 55, 188
- AUD 240
- audio output jack 2, 7, 240, 346
- auto-answer operation 167
- auto-eject feature 129
- automatic repeat 3
- auxiliary RAM 20

B

- B command 145, 157
- back panel 8–9
- backslash (\) 55, 58
- backspace 58–59
- BadCmd \$01 error 130, 131
- bank selector switches 25–33
- bank-switched memory 22–23
- BASIC
 - and analog inputs 186
 - and serial port 1 145
 - and switch inputs 185
 - Applesoft 317
 - Integer 317
 - prompt character for 55
 - restarting from 113
 - support for mouse 182
- BASIC program(s)
 - and mouse 175
 - and MouseText 83
 - exiting from 63
 - invoking Monitor from 188
 - returning to from Monitor 188
- baud rate(s) 152, 261, 262
 - setting 145, 157
- Bell routine 78
- Bell1 routine 78
- bit values 374–375
- bit-mapping 89
- black-and-white monitor, and colors 87, 89–90
- blanking intervals 241
- BREAK character 147, 159
- BREAK signal 152
- BRK (\$00) instruction(s) 80
 - handling 325
- BRK vector 70

- buffering
 - keyboard 329
 - serial 331
- buffer(s)
 - alignment 59
 - display 34, 36
 - input 34, 36
 - pointer 58
- built-in disk drive 8
- built-in firmware routines, calling 52
- BusErr \$06 error 116
- button interrupt mode 176
- bytes 375

C

- C command 146, 158
- C flag 116, 141
- C3COut1 routine 59, 61
- C3KeyIn routine 54
- cables, attaching 8
- CALL -151 command 112, 188
- Canadian keyboard 363
- cancel line 58
- [CAPS LOCK]** key 5
- carriage return 146, 153, 158, 166
- carrier 152
- CH (cursor horizontal) 60
- character modifier keys 76
- character output switch (CSW) 53
- character set(s)
 - alternate 65, 82
 - primary 65, 82
 - specifications 81
- characters
 - control 5
 - dimensions of 82
 - flashing 64, 82
 - inverse 64, 82
 - MouseText 65, 83
 - normal 64, 82
- CIEOLZ routine 105
- ClrEOL routine 105
- ClrEOP routine 105
- ClrScr routine 105
- ClrTop routine 105
- clamping boundaries, mouse 182
- ClampMouse routine 180
- CLC instruction 41
- ClearMouse routine 180
- clock rate 220
- clock signals 223-225
- clock source 261, 262
- CLOSE call 132
- cold-start reset 48-49, 130
- cold-start routine 113
- colors
 - double-high-resolution 92
 - high-resolution 90, 251-252
 - low-resolution 88, 250
- column-address strobe (CAS) 236
- command characters
 - changing 147, 160
 - port 1 (CONTROL-I) 145, 147
 - port 2 (CONTROL-A) 157, 160
 - substitute 147, 160
- command register, ACIA 263
- connector slots, peripheral 66
- connectors 8-9
 - disk drive 257
 - external power 218
 - hand controller 270
 - mouse 267
 - serial port 261
 - video output 254
- CONTINUE BASIC command 199

- CONTROL call 127-129
- control characters 5
 - disabling 57
 - reactivating 57
 - sending without interception 154
 - with C3COut1 61, 65
 - with COut1 60, 65
- [CONTROL]** key 4, 5
- control register, ACIA 261-262
- CONTROL-} or | 62, 110
- CONTROL-| or \ 62, 110
- CONTROL-[62
- CONTROL-^ or 6 110
- CONTROL-__ 62, 110
- CONTROL-A 157, 160, 166
- CONTROL-B 199
- CONTROL-C 63, 199
- CONTROL-D 145, 157
- CONTROL-E 109
- CONTROL-F 109
- CONTROL-G 60, 61, 77, 78, 109
- CONTROL-H 58, 60, 61, 109
- CONTROL-I 145, 147
- CONTROL-J 60, 61, 109
- CONTROL-K 61, 109, 199
- CONTROL-L 61, 109
- CONTROL-M 60, 61, 109
- CONTROL-N 61, 84, 109
- CONTROL-O 61, 83, 109
- CONTROL-P 199
- CONTROL-Q 61, 80
- CONTROL-R 61, 159
- CONTROL-RESET 49, 188
- CONTROL-S 61
- CONTROL-T 159
- CONTROL-U 62, 76
- CONTROL-V 62, 109, 147, 160
- CONTROL-W 62, 109, 147, 160

- CONTROL-X 58, 62, 76
- CONTROL-Y 62, 109, 202
- CONTROL-Z 62, 110
- COut routine 59–60, 106
- COut1 routine 59, 60, 106
- CPU 12
- CR (carriage return) 153
- CROut routine 106
- CROut1 routine 106
- CSW link 53
- cursor 4
 - blinking underscore 166
 - block 54
 - checkerboard 54
 - flashing checkerboard 52
 - flashing question mark 145, 157
 - inverse plus sign 58
 - inverse solid 52
 - movement codes 56–57, 60, 61
 - movement keys 4, 58
 - position 59–60
- CV (cursor vertical) 60

D

- D command 146, 158
- data format, setting 146, 158
- data inputs 21
- Data Terminal Ready, setting levels for 263
- data transfer 261
- DB-9 connector 175, 184–185, 266, 270
- DB-15 connector 81, 254
- DB-19 connector 256–257
- debugging programs 205–207
- DELETE** key 4
- device control block (DCB) 121, 129
- device identification bytes 67

- device information block (DIB) 121–122
- DHiRes switch 44, 96
- DIN jacks 260
- disk controller unit (IWM) 231
- disk drive 8
 - I/O firmware entry points 21
 - speed adjustment 13
- disk-use light 7
- display
 - double-high-resolution 252–253
 - high-resolution 251
 - low-resolution 250
- display address transformation 243
- display format(s) 82, 198
 - inverse 61
 - normal 61
 - specifications 81
- display memory
 - addressing 242
 - maps 98–99
 - switches 43–44
- display mode specifications 81
- display page(s)
 - addressing 98–99
 - HRP1 36, 93
 - HRP1X 36, 93
 - HRP2 37, 93
 - HRP2X 37
 - locations 94
 - switches for selecting 43–44
 - TLP1 34, 93
 - TLP1X 36, 93
 - TLP2 36, 93
 - TLP2X 36
- display soft switches 95–96
- DisVBI switch 177
- DisXY switch 177

- DOS 316
- double-high-resolution graphics 91, 103
 - generating 252–253
 - specifications 81
- double-low-resolution graphics 94
- ↓** key 4, 153
- Dvorak keyboard 5–6, 358
- dynamic RAM refreshment 235
- dynamic RAM timing 236–237

E

- echoing 263
- editing features
 - backspace 58
 - cancel line 58
 - retype 59
- 80Col switch 95
- 80-column display 84, 93
 - address mapping 242–243
 - addressing 247
 - and CH value 60
 - and Monitor 80
 - left margin value 63
 - map of 100
 - switching to 5, 57, 61
 - timing signals 249
 - width value 63
- 80/40 switch 5, 80
- 80Store switch 43, 95
- EnbXY switch 177
- English keyboard 360
- enhanced video firmware 64, 65, 69, 80
- entry points
 - firmware 20–21
 - for ports 66
 - Monitor 313–314

EnVBI switch 177
error codes, Protocol Converter
 141-142
[ESC] key 56
escape mode 56-58
even parity 152
EXAMINE command 197
external drive startup 114
EXTINT line 121

F

F command 146, 158
firmware
 addresses 311-313
 entry points 20-21, 66
 listings 385-470
 mapping 64-65
 protocol 67, 108, 162, 181
firmware routines
 calling 52
 input 54-55
 mouse 180
 output 59
 video 104-110
flag inputs 21
flags 16, 21
flashing characters 64, 82
flashing power light 7
flashing question mark 145, 157
folded address bits 244
forced cold-start reset 48-50
FORMAT call 126
FORTRAN 317
40-column display
 address mapping 242-245
 addressing 247
 left margin value 63

map of 99
memory locations 245
switching to 5, 57, 61
timing signals 248
width value 63
with double-high-resolution graphics
 91
48K memory 34-35
 switches 37
French keyboard 361
full-duplex operation 168-169

G

game input characteristics 184
general logic unit (GLU) 230
German keyboard 364
GetLn routine 55-56
GetLn1 routine 56, 76
GetLnZ routine 76
GLU (general logic unit) 230
GO command 203
graphics display
 double-high-resolution 91
 double-low-resolution 94
 high-resolution 88-89
 low-resolution 87
graphics routines 37
grounding 217, 371

H

half-duplex operation 167-168
hand controller(s)
 and mouse 183-184
 circuits 272
 connector 270, 271
 signals 185-186, 270, 273

handle 8
headphone jack 2, 7, 240, 346
hexadecimal
 arithmetic 199
 converting to decimal 376
 converting to negative decimal 377
high-resolution graphics 88-89
 display map 102
 generating 251
 specifications 81
 storing in memory 89
HiRes switch 43, 96
HLine routine 106
HOME routine 106
HomeMouse routine 180
horizontal-count bits 244
HRP1 36, 93
HRPIX 36, 93
HRP2 37, 93
HRP2X 37

I

I command 146, 158
I/O firmware entry points 20-21
I/O ROM space 69
I/O subroutines, calling 52
identification (ID) bytes
 Apple II series 338
 device 67
IN#2 command 156, 157, 166
IN#6 command 112
IN#n command 53, 66
index register(s) 16, 202
indirect addressing 24
INIT call 130
InitMouse routine 180
input buffer 34

- input subroutines
 - GetLn 55–56
 - GetLn1 76
 - GetLnZ 76
 - KeyIn 54
 - RdChar 76
 - RdKey 54
- input/output unit (IOU) 227–228
- inputs
 - analog 186
 - data 21
 - flag 21
 - switch 185
- Integer BASIC, prompt character for 55
- integrated circuits 12
- Integrated Woz Machine (IWM) 231
- internal voltage converter 219–220
- interrupt handler, Monitor 70
- interrupt(s) 70, 223
 - allowing 263
 - Apple IIc support for 320–321
 - causes of 321
 - handling 322–324, 327–332
 - sources of 326
 - VBInt 178
 - XInt 178
 - YInt 178
- inverse characters 64, 82
- INVERSE command 83, 198
- inverse flag 64, 65
 - values 65
- IOU (input/output unit) 227–228
- IOUdis switch 44, 95, 96, 177
- IRQ vector 70
- ISO keyboard layout 359
- Italian keyboard 366
- IWM (Integrated Woz Machine) 231


J

- jack(s)
 - audio output (headphone) 2, 7, 240, 346
 - DIN 260
 - phono 81
- JMP indirect instruction 283
- joysticks *See* hand controller(s)
- jump table, constructing 68

K

- K command 146, 158
- keyboard 3
 - characteristics 73
 - circuit diagram 238
 - disabling 146, 158
 - enabling 146, 158
 - encoder 72
 - features 3
 - interrupts from 328–330
 - layout(s) 3, 5–6, 354–369
 - operation of 237–238
 - signals 239
 - specifications 4
 - switch 5
- keyboard input switch (KSW) 53
- keyboard strobe
 - clearing 72–74
 - reading 72
- KeyIn routine 54–55
- keys
 - cursor movement 4
 - modifier 5, 76
 - special function 4
- KSW link 53

L

- L command 146
- last opened location 189
-  key 4, 58
- line feed (LF) 153, 166
 - disabling 146, 158
 - enabling 146, 158
 - masking 146, 158
- line width 153
 - setting 145, 146, 157, 158
- LIST command 204–205, 210
- Logo II 318
- loudspeaker 7
- low-resolution graphics 87
 - display map 101
 - generating 250
 - specifications 81
 - storing in memory 87



M

- M command 146, 158
- machine-language programs 203–205
- main RAM 20
- maps
 - Apple IIc memory 19
 - bank-switched memory 23
 - double-high-resolution graphics display 103
 - 80-column text display 100
 - 40-column text display 99
 - 48K memory 35
 - high-resolution graphics display 102
 - low-resolution graphics display 101
- MARK parity 152

- memory
 - addressing 232
 - bank-switched 22-23
 - changing contents of 192-194
 - comparing data in 196
 - displaying contents of 190-192
 - dump 190-192
 - 48K 34-35
 - map 19
 - moving data in 194-195
 - range 191
 - memory address(es) 18
 - controlling text window 63
 - for data I/O 68
 - for display modes 245
 - hardware 21-22
 - hardware page 303-307
 - of text displays 247
 - page \$00 294-297
 - page \$03 298
 - port screen hole 69-70
 - RAM 20, 234-237
 - ROM 20, 233-234
 - transforming display data into 242
 - memory bus organization 233
 - memory management unit (MMU) 225-226
 - memory pages
 - \$00 (zero page) 18, 24
 - \$01 (stack) 18, 24
 - \$02 (input buffer) 34
 - \$03 (global storage, vectors) 34
 - \$04-\$07 (TLP1) 34
 - \$08 (communication port buffers) 36
 - \$08-\$0B (TLP2) 36
 - \$20-\$3F (HRP1) 36
 - \$40-\$5F (HRP2) 37
 - \$D0-\$FF (ROM and RAM) 24
 - memory switches
 - addresses of 21-22
 - bank selector 25-33
 - display 43-44
 - 48K 37
 - hardware functions of 21-22
 - microprocessor 12, 16 *See also* 65C02
 - microprocessor
 - Mini-Assembler 207
 - instruction formats 210-211
 - leaving 210
 - starting 208
 - using 209-210
 - MIXED switch 91, 95
 - mixed-mode displays 91
 - MMU (memory management unit) 225-226
 - mode bytes 179
 - modem, baud rate of 262
 - modem communication 165
 - modem port *See* port 2
 - monitor
 - black-and-white 80, 87, 89
 - color 80, 87, 89-90
 - television set as 80, 84, 87, 89-90
 - Monitor
 - firmware addresses 21
 - interrupt handling by 70
 - keyboard support 72
 - memory location of 188
 - prompt character for 55
 - Monitor commands
 - for debugging 205-207
 - for machine-language programs 203
 - memory commands 189-196
 - register commands 197
 - repeating 201-202
 - syntax of 189
 - Monitor subroutines
 - for game input 186
 - for input 54, 76
 - for output 59, 78
 - for video display 104-110
 - mouse
 - circuits 268
 - disabling 182
 - enabling 179
 - firmware entry points 21
 - firmware routines 180
 - interrupt handler 178
 - interrupts 333
 - movement signals 176
 - operation of 265
 - port characteristics 174
 - screen hole locations 182
 - soft switches 177-178
 - waveforms 266
 - MouseText graphics characters 65, 83
 - and ASCII characters 84
 - disabling 62
 - printing 84
 - MouXI switch 178
 - MouYI switch 178
 - MOVE command 194, 200-201
 - MoveAux routine 40-41
 - movement interrupt mode 175-176
 - movement/button interrupt mode 176
 - multiplexing 235
 - RAM address 236
- ## N
- N command 146, 158
 - n CONTROL-K 53
 - n CONTROL-P 53
 - newline
 - enabling or disabling 129
 - status 121

next changeable location 189
nibbles 375
normal characters 64, 82
NORMAL command 84, 198, 200
NTSC 90, 241
#6 command 145

O

odd parity 152
1 CONTROL-P command 144
 key 4, 50, 76
 **CONTROL** **RESET** 49–50
OPEN call 131
operating systems 53
output jack 240
output subroutines
 Bell 78
 Bell1 78
 COut 59–60
 COut1 60
 C3COut1 61
 for video display 104–110
overheating 216

P

P command 147, 159
P register 16
paddle values 274
Page 1
 high-resolution (HRP1) 36, 93
 text and low-resolution (TLP1) 34, 93
Page 1X
 high-resolution (HRP1X) 36, 93
 text and low-resolution (TLP1X) 36, 93

Page 2
 high-resolution (HRP2) 37, 93
 text and low-resolution (TLP2) 36, 93
Page 2X
 high-resolution (HRP2X) 37
 text and low-resolution (TLP2X) 36
Page2 switch 43, 95
pages, display *See* display pages
pages, memory *See* memory pages
PAL 241
parity, setting 147, 159
parity bit 152
parity checking 263
Pascal 317
 and external drive startup 114
 and I/O links 53
 and printer output 145
 and window widths 63
 firmware protocol 67
 ID byte 149, 162, 181
 operating system 316
 support for mouse 181
 transferring files to modem with 157

PC (program counter) 16
Pd10 186
Pd11 186
PEEK xxxi, 185, 317
peripheral connector slots 66
peripheral identification numbers (PIN) 378–379
PInit routine 67, 108
 entry point of 149, 162

pinouts
 GLU 230
 IOU 227
 IWM 231
 MMU 226
 6551 260
 TMG 229
 video expansion connector 255
PLOT routine 106
POKE 317
port 1
 characteristics 144
 characteristics storage 151
 commands 145–147
 configuration at startup 148
 firmware protocol 149
 hardware addresses 149
 screen hole locations 150
port 2
 characteristics 156
 characteristics storage 163
 commands 157–159
 configuration at startup 160
 firmware protocol 162
 hardware addresses 161
 screen hole locations 162
port routine requirements 68
ports
 serial port 1 20
 serial port 2 20
PosMouse routine 180
power light 7
power supply 10–11, 217–218, 219
 50-Hz 372
power switch 8
power-on reset 48–49
power-up byte 50
PR#1 command 144, 145, 156
PR#2 command 156, 157
PR#3 command 83

- PR#6 command 112, 113
- PR#n command 53, 66
- PrBl2 routine 107
- PrByte routine 107
- PRead routine 67, 109, 186
 - entry point of 149, 162
- PrErr routine 107
- PrHex routine 107
- primary character set 64, 65, 82
- PRINT CHR\$(24) 83
- PRINT CHR\$(27) 83
- PRINTER command 145
- printer port *See* port 1
- PrntAX routine 107
- processor 12
- processor status register 16
- ProDOS 316
- program counter (PC) 16, 208
- prompt characters 55
- protocol, firmware 67
 - for mouse port 181
 - for port 1 149
 - for port 2 162
 - for port 3 108
- Protocol Converter
 - calls to 117–118
 - entry point 115
- PStatus routine 68, 110
 - entry point of 149, 162
- PTrig switch 177
- PWrite routine 67, 109
 - entry point of 149, 162

Q

- Q command 159
- question mark (?) 55
- quitting terminal mode 159

R


- R command 147, 159
- RAM address(es) 20
 - multiplexing 236
- RAM addressing 234–237
- RAMRd switch 37
- RAMWrt switch 37
- random-number routine 55
- RCA-type phono jack 81
- Rd80Col switch 95
- Rd80Store switch 43, 95
- RdAltChar switch 95
- RdAltZP switch 26
- RdBnk2 switch 26
- RdBtnO switch 178
- RdChar routine 76
- RdDHiRes switch 44, 96
- RdHiRes switch 43, 96
- RdIOUDis switch 44, 96, 177
- RdKey routine 54
- RdLCRAM switch 26
- RdMIXED switch 95
- RdPage2 switch 43, 95
- Rd63 switch 178
- RdTEXT switch 95
- RdVBIMsk switch 177
- RdXOEdge switch 177
- RdXYMsk switch 177
- RdYOEdge switch 177
- READ BLOCK call 123–124
- READ call 133–134
- ReadMouse routine 180
- receiver clock source 261, 262
- registers 16
 - changing 197
 - examining 197
- remote mode 171

Request To Send (RTS) 263

- RESET** key 4, 47, 76
- reset routine 47–48
- reset vectors 48–50
- reset(s)
 - cold-start (power-on) 49, 130
 - forced cold-start 49–50
 - warm-start 49, 128
- RETURN** key 4
- RF video modulator 80
- ROM addresses 20–21
- ROM addressing 233–234
- row-address strobe (RAS) 236
- RstVBI switch 177
- RstXInt switch 177
- RstXY switch 177
- RstYInt switch 177
- RTS (Request To Send) 263

S

- S command 147, 159
- S register 16
- safety instructions 217, 371
- schematic diagrams 274–280
- screen alignment 59
- screen hole locations 36, 69–70, 298
 - in auxiliary memory 301
 - in main memory 299–300
- mouse port 182
 - port 1 150
 - port 2 162
- SCRN routine 107
- scrolling 60–62
- SEC instruction 41
- serial port 1 20
- serial port 2 20
- serial port circuits 258

- ServeMouse routine 175, 180
 - SetCol routine 107
 - SetMouse routine 179, 180
 - SetPWRG routine 50
 - [SHIFT]** key 5
 - Sholes keyboard 5-6, 355
 - 6 CONTROL-K command 112
 - 6 CONTROL-P command 83, 112, 113
 - 65C02 microprocessor 12
 - block diagram of 220
 - data sheet for 283-292
 - registers 16
 - specifications 223
 - versus 6502 282-283
 - signals
 - character-generator control 250
 - disk drive connector 257
 - external power connector 218
 - 14-MHz video timing 249
 - GLU 230
 - hand controller 273
 - hand controller connector 271
 - IOU 227
 - IWM 231
 - keyboard 239
 - memory selection 233
 - MMU 226
 - mouse button 269
 - mouse connector 267
 - RAM timing 236-237
 - serial port connector 261
 - 7-MHz video timing 248
 - 65C02 timing 223-225
 - 6551 260
 - TMG 229
 - video output 253-256
 - slot equivalents 66
 - soft switches 18, 22
 - bank selector 25
 - display 95-96
 - display memory 43, 44
 - 48K memory 37
 - hardware functions of 21-22
 - memory addresses of 21-22
 - mouse 177-178
 - resetting to normal 47-48
 - speaker 77
 -  4, 50, 76
 - SPACE parity 152
 - speaker
 - characteristics 77
 - operation of 240
 - toggle 22
 - using 77
 - volume control 7, 240
 - special function keys 4
 - specifications
 - environmental 216-217
 - 50-Hz power supply 372
 - internal converter 219
 - keyboard 4
 - power supply 218
 - 65C02 microprocessor 223
 - video display 81
 - speed adjustment, disk drive 13
 - stack 18, 24
 - stack pointer 16
 - standard (Sholes) keyboard 5-6
 - start bit 152
 - startups
 - cold-start routine 113
 - from external drive 114
 - warm-start routine 113
 - STATUS call 119-123
 - status register, ACIA 264
 - STEP command 205-207
 - stop-list function 61, 62-63
 - strobe
 - inputs 21
 - outputs 22
 - Super Serial Card 144, 156, 349
 - Sw0 185
 - Sw1 185
 - switch inputs 185, 270
 - switches
 - address space (memory) 21-22
 - bank selector 25-33
 - character output 53
 - keyboard input 53
 - soft 18, 21-22
 - sync pulses 241
 - synchronization signals 241
- T
- T command 159
 - [TAB]** key 4
 - television set
 - and colors 87, 89-90
 - and display width 84
 - and high-resolution graphics 90
 - as monitor 80
 - vertical line count and 242
 - terminal mode 159, 166, 171
 - text capacity specifications 81
 - text display(s)
 - characteristics 86
 - 80-column 5, 80, 84, 93
 - 40-column 5, 80, 84, 93
 - generating 247
 - switching 86
 - text modes 82
 - TEXT switch 91, 95
 - text window
 - changing 63
 - values for parameters 64

timing generator (TMG) 229
timing signals 223–225
TLP1 34, 93
TLP1X 36, 93
TLP2 36, 93
TLP2X 36
TMG (timing generator) 229
TMG chip 262
toggle switches 22
TRACE command 205–207
transmit/receive register, ACIA 265
transparent mode 175
2 CONTROL-P command 156
two-key rollover 113

U

UniDisk 3.5 112
 and mouse interrupts 179
 and slot 7 348
 cold starts with 50
 communicating with 114–115
 debugging commands 205–207
 ROM size 233–234
 starting up from 113, 114
USER command 202

V

validity-check byte 49, 50
VBIInt signal 175, 178
vectors, Monitor 313–314
ventilation 8
VERIFY command 196, 201
vertical blanking active modes 176
vertical-count bits 244
video control functions 109–110
video counters
 horizontal 241–242
 vertical 242
video display
 operation of 241
 specifications 81
video output firmware entry points 20
video port characteristics 80
video signal 81
VLine routine 108
voltage converter 10–11
voltages
 50-Hz power supply 372
 internal converter 219–220
 power supply 218
 65C02 microprocessor 223
volume control 7, 240
VTab 60

W

warm-start reset 49, 128
warm-start routine 113
Western Spanish keyboard 368
WRITE BLOCK call 124–125
WRITE call 134–135
write-enable addresses 25

X

X command 147, 159
X register 16
 and port routines 68
X1 266
XFer routine 40–42
XInt 178
X0 176, 265
X0Edge switch 177
XON/XOFF protocol 147, 159

Y

Y register 16
 and port routines 68
Y1 266
YInt 178
Y0 176, 265
Y0Edge switch 177

Z

Z command 147, 154, 159
zapping command characters 147, 154,
 159
zero page 18, 24

